



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

© 1998-2010 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: May 2010

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
SDK for Enterprise Architect	2
Developing Profiles	3
Custom Stereotypes	3
Create Profiles	5
Create a Profile Package	5
Add Stereotypes and Metaclasses	6
Define Stereotype Tagged Values	8
With Predefined Tag Types	9
With Supported Attributes	10
Use the Tagged Value Connector	11
Define Stereotype Constraints	12
Add Enumeration Elements	14
Add Shape Scripts	15
Set Default Appearance	17
Export a UML Profile	17
Save Profile Options	18
Supported Attributes	19
Define a Stereotype as a Metatype	21
Define Multiple-Stereotype Level	21
Define Creation of Instance	22
Create Composite Elements	23
Define Child Diagram Types	23
Stereotype Profiles	24
Quick Linker	25
Quick Linker Definition Format	25
Quick Linker Example	27
Hide Default Quick Linker Settings	28
Quick Linker Object Names	29
MDG Technologies in SDK	30
Create MDG Technologies	30
Add a Profile	35
Add a Pattern	35
Add a Diagram Profile	36
Add a Toolbox Profile	37
Add Task Panel Pages	38
Add Tagged Value Types	39
Add Code Modules	40
Add MDA Transforms	42
Add Images	43
Add Scripts	43
Add RTF Report Templates	44
Add Linked Document Templates	45
Working with MTS Files	46
Customize Toolbox Profiles	47
Create Toolbox Profiles	47
Toolbox Page Attributes	48
Create Hidden Sub-Menus	48
Override Default Toolboxes	49
Assign Icons To Toolbox Items	49
Enterprise Architect Toolboxes	50
Elements Used in Toolboxes	50

Connectors Used In Toolboxes	51
Create Diagram Profiles	52
Built-In Diagram Types.....	53
Attribute Values - stylex & pdata	53
Create Tasks Pane Profiles	54
Define Tasks Pane Toolboxes.....	55
Built-In Tasks Pane Commands.....	55
Run Add-In Functions.....	56
Define Tasks Pane Contexts.....	57
Allocate Tasks Pane Contexts.....	57
Save a Tasks Pane Profile.....	58
Define Validation Configuration	58
Incorporate Model Templates	59
Deploy An MDG Technology	59
Shape Scripts	61
Getting Started With Shape Scripts	61
Shape Editor	64
Write Scripts	65
Syntax Grammar.....	65
Shape Attributes.....	66
Drawing Methods.....	68
Color Queries.....	72
Conditional Branching.....	72
Query Methods.....	72
Display Item Properties.....	72
Sub-Shapes.....	75
Reserved Names.....	76
Miscellaneous.....	76
Example Scripts	77
Tagged Value Types	81
Predefined Structured Types	81
Create Structured Tagged Values	83
Predefined Reference Data Types	84
Create Reference Data Tagged Values	85
Create Custom Tagged Value Type	86
Code Template Framework in SDK	88
Code Template Syntax	88
Literal Text.....	88
Macros.....	89
Template Substitution Macros	89
Field Substitution Macros.....	90
Tagged Value Macros.....	102
Function Macros.....	103
Control Macros.....	106
EASL Code Generation Macros.....	109
EASL Collections.....	111
EASL Properties.....	113
Variables.....	117
The Code Template Editor in SDK	118
Custom Templates.....	119
Override Default Templates.....	120
Add New Stereotyped Templates.....	121
Create Custom Language Template	122
Enterprise Architect Add-In Model	123
Add-In Tasks	124
Create Add-Ins.....	124
Define Menu Items.....	124

Deploy Add-Ins.....	125
Tricks and Traps.....	126
The Add-In Manager	128
Add-In Search	128
XML Format (Search Data).....	129
Add-In Events	129
EA_Connect.....	130
EA_Disconnect.....	130
EA_GetMenuItems.....	131
EA_GetMenuState.....	131
EA_MenuClick.....	132
EA_OnOutputItemClicked.....	133
EA_OnOutputItemDoubleClicked.....	133
EA_ShowHelp.....	134
Broadcast Events	135
EA_FileOpen.....	135
EA_FileClose.....	136
EA_FileNew.....	136
EA_OnPostCloseDiagram.....	137
EA_OnPostOpenDiagram.....	137
Pre-Deletion Events.....	138
EA_OnPreDeleteElement.....	138
EA_OnPreDeleteAttribute.....	138
EA_OnPreDeleteMethod.....	139
EA_OnPreDeleteConnector.....	139
EA_OnPreDeleteDiagram.....	140
EA_OnPreDeletePackage.....	141
Pre-New Events.....	141
EA_OnPreNewElement.....	141
EA_OnPreNewConnector.....	142
EA_OnPreNewDiagram.....	143
EA_OnPreNewDiagramObject.....	143
EA_OnPreNewAttribute.....	144
EA_OnPreNewMethod.....	145
EA_OnPreNewPackage.....	145
EA_OnPreExitInstance.....	146
Post-New Events.....	146
EA_OnPostNewElement.....	147
EA_OnPostNewConnector.....	147
EA_OnPostNewDiagram.....	148
EA_OnPostNewDiagramObject.....	148
EA_OnPostNewAttribute.....	149
EA_OnPostNewMethod.....	149
EA_OnPostNewPackage.....	150
EA_OnPostInitialized.....	151
EA_OnPostTransform.....	151
Technology Events.....	152
EA_OnInitializeTechnologies.....	152
EA_OnPreActivateTechnology.....	152
EA_OnPostActivateTechnology.....	153
EA_OnPreDeleteTechnology.....	153
EA_OnDeleteTechnology.....	154
EA_OnImportTechnology.....	155
Context Item Events.....	155
EA_OnContextItemChanged.....	155
EA_OnContextItemDoubleClicked.....	156
EA_OnNotifyContextItemModified.....	157
Compartment Events.....	158

EA_QueryAvailableCompartments.....	158
EA_GetCompartmentData.....	159
Model Validation Broadcasts.....	160
EA_OnInitializeUserRules.....	160
EA_OnStartValidation.....	161
EA_OnEndValidation.....	161
EA_OnRunElementRule.....	162
EA_OnRunPackageRule.....	162
EA_OnRunDiagramRule.....	162
EA_OnRunConnectorRule.....	163
EA_OnRunAttributeRule.....	163
EA_OnRunMethodRule.....	164
EA_OnRunParameterRule.....	164
Model Validation Example.....	165
EA_OnRetrieveModelTemplate.....	168
Custom Views	169
Create a Custom View.....	169
MDG Add-Ins	170
MDG Events.....	170
MDGBuild Project.....	171
MDGConnect.....	171
MDGDisconnect.....	172
MDGGetConnectedPackages.....	173
MDGGetProperty.....	173
MDGMerge.....	174
MDGNewClass.....	175
MDGPostGenerate.....	176
MDGPostMerge.....	176
MDGPreGenerate.....	177
MDGPreMerge.....	177
MDGPreReverse.....	178
MDGRunExe.....	179
MDGView.....	179
Enterprise Architect Object Model	181
Using the Automation Interface	181
Connect to the Interface.....	181
Set References In Visual Basic.....	183
Examples and Tips.....	184
Call from Enterprise Architect.....	185
Available Resources.....	186
Reference	186
Interface Overview.....	187
App.....	190
Enumerations.....	191
ConstLayoutStyles Enum.....	191
CreateBaselineFlag Enum.....	192
CreateModelType Enum.....	192
EAEditionTypes Enum.....	192
EnumRelationSetType Enum.....	192
ExportPackageXMIFlag Enum.....	193
MDGMenus Enum.....	193
ObjectType Enum.....	193
PropType Enum.....	194
ReloadType Enum.....	194
ScenarioDiagramType Enum.....	194
ScenarioStepType Enum.....	195
ScenarioTestType Enum.....	195
XMIType Enum.....	195

Repository.....	196
Repository.....	197
Author	210
Client	210
Collection	211
Datatype	213
EventProperties.....	214
EventProperty.....	214
ModelWatcher.....	214
Package	215
ProjectIssues.....	219
ProjectResource.....	220
PropertyType.....	221
Reference.....	222
Stereotype.....	223
Task	223
Term	224
Element.....	225
Constraint.....	227
Effort	227
Element	228
File	235
Issue (Maintenance).....	235
Metric	236
Requirement.....	237
Resource	238
Risk	239
Scenario	239
ScenarioExtension.....	240
ScenarioStep.....	241
TaggedValue.....	242
Test	243
Element Features.....	243
Attribute	244
AttributeConstraint.....	246
AttributeTag.....	247
CustomProperties.....	248
EmbeddedElements.....	248
Method	249
MethodConstraint.....	251
MethodTag.....	252
Parameter.....	252
Partitions	253
Properties.....	254
Transitions.....	255
Connector.....	255
ConnectorConstraint.....	256
Connector.....	257
ConnectorEnd.....	260
ConnectorTag.....	262
RoleTag	263
Diagram.....	263
Diagram	264
DiagramLinks.....	267
DiagramObjects.....	268
SwimlaneDef.....	269
Swimlanes.....	270
Swimlane	271

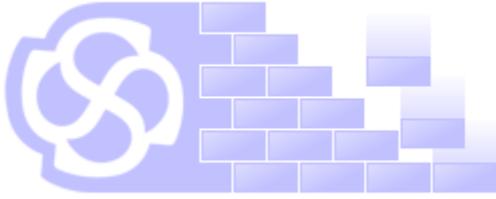
Project Interface.....	271
Project	271
Code Samples.....	284
Open the Repository.....	284
Iterate Through a .EAP File.....	284
Add and Manage Packages.....	285
Add and Manage Elements.....	286
Add a Connector.....	286
Add and Manage Diagrams.....	287
Add and Delete Features.....	288
Element Extras.....	288
Repository Extras.....	290
Stereotypes.....	292
Work With Attributes.....	292
Work With Methods.....	293

Index**295**

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

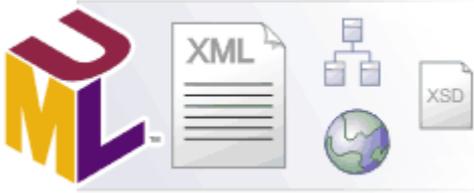
Welcome to the *Enterprise Architect Software Developers Kit (SDK)*. This is a special section of the *Enterprise Architect User Guide*, covering the more advanced aspects of extending and customizing Enterprise Architect.

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](#)^[3] (incorporating [UML Stereotypes](#)^[3])
- [MDG Technologies](#)^[30]
- [Shape Scripts](#)^[61]
- [Tagged Value Types](#)^[81]
- [Code Template Framework](#)^[88]
- [Enterprise Architect Object Model \(Automation Interface\)](#)^[181]
- [Enterprise Architect Add-In Model](#)^[123]

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](#)^[3] and [Tagged Values](#)^[81] 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](#)^[3]
- [Create Profiles](#)^[5]
- [Quick Linker](#)^[25]
- [Toolbox Profiles](#)^[47]
- [Diagram Profiles](#)^[52]
- [Task Pane Profiles](#)^[54]

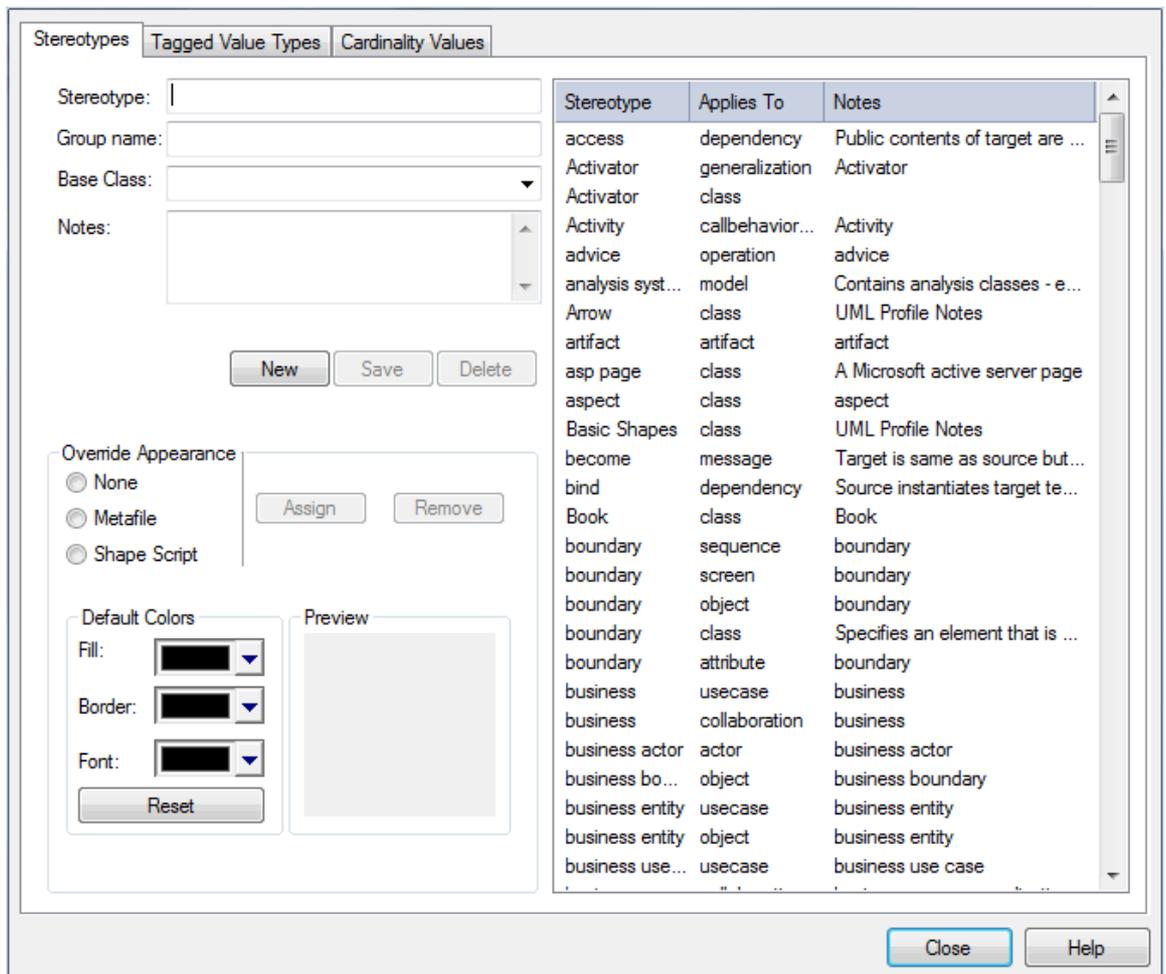
1.1 Custom Stereotypes

UML supports a large number of *stereotypes*, which are an inbuilt mechanism for logically extending or altering the meaning, physical appearance 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
Stereotype	Specify the name of the stereotype.
Group name	Enable grouping of stereotype features by a plural name, for attributes and operations, which is shown on diagrams in the attribute and operations compartments.
Base Class	Enable the stereotyped element to inherit the base characteristics from a pre-existing element type.
Notes	Type any notes concerning the stereotype (not the elements to which the stereotype is to be applied).
Override Appearance	
None	Switch to the default element appearance.
Metafile	Enable an image file to be used for the appearance of the stereotype.
Shape Script	Specify custom shapes for the stereotype using the <i>Enterprise Architect Shape</i>

Option	Use to
	Scripting language. For more information see the Shape Scripting topic.
Assign	Add the associated metafile or Shape Script from the stereotyped element.
Remove	Remove the associated metafile or Shape Script from the stereotyped element.
Default Colors	
Fill	Set the default background color of the element.
Border	Control the border color.
Font	Control the color of the stereotype font.
Reset	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.

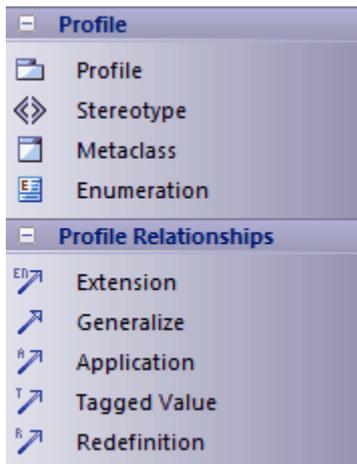
To create a UML Profile, follow the steps below:

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

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)**.

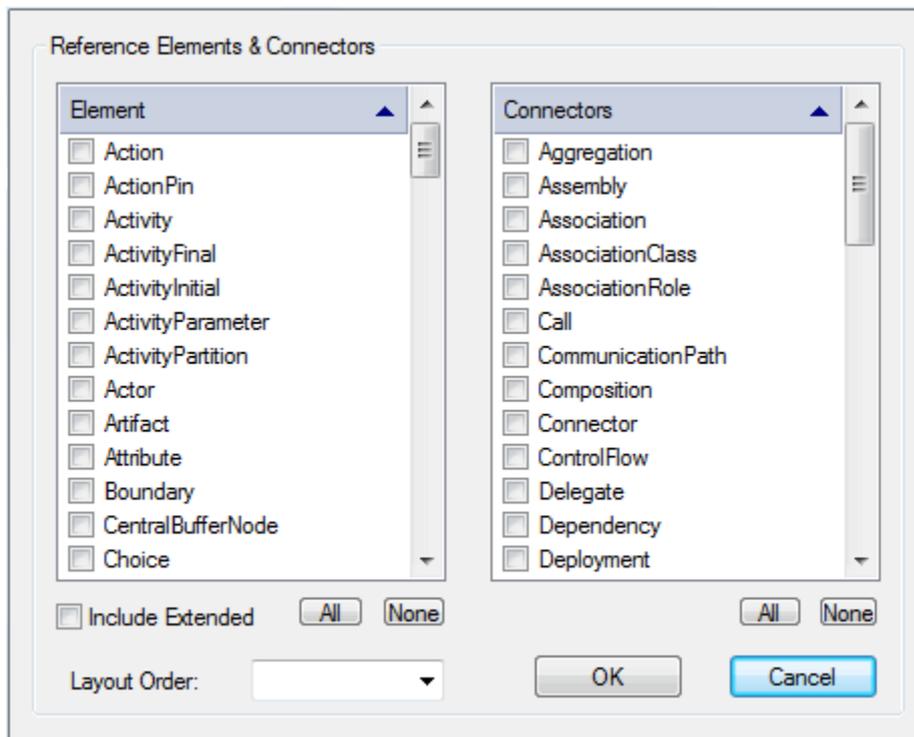


3. Drag the *Profile* item onto the Class diagram. The **New Model Package** dialog displays.
 4. In the **Package Name** field, type a name for the Profile.
 5. Select the **Automatically add new diagram** checkbox.
 6. Click on the **OK** button. The **New Diagram** dialog displays.
 7. Provide the 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](#)  to the Profile.

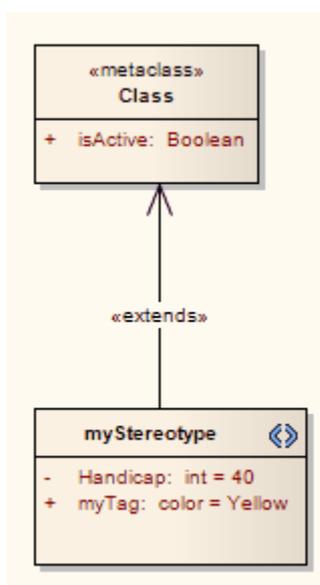
1.2.2 Add Stereotypes and Metaclasses

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.

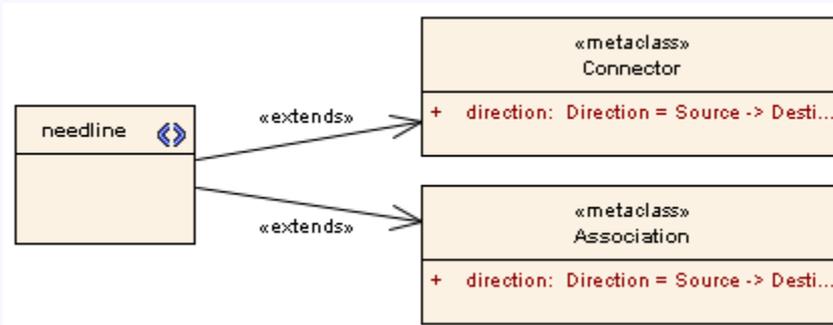


3. Scroll down the **Element** list and select the checkbox for **Class**.
4. Click on the **OK** button, display the Class **Properties** dialog, and in the **Name** field type a name for the element. Click on the **OK** button again.
5. Drag a **Stereotype** element from the **Toolbox** onto the diagram. 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:



Note:

If you want to have a stereotype extending more than one metaclass, do not create two stereotype Classes with the same name. You cannot have two stereotypes with the same name in the same profile; one is discarded when you save the profile. Therefore, create one stereotype Class with an Extension connector to each of several Metaclass elements, as shown below.



You can now add [stereotype Tags](#)^[8], [Constraints](#)^[12], [Enumerations](#)^[14], and/or [Shape Scripts](#)^[15] to your Profile, and define the [default appearance](#)^[17] 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](#)^[9]
- [Define Stereotype Tags with Supported Attributes](#)^[10]
- [Use the Tagged Value Connector](#)^[11]

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

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

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

- General Tab:**
 - Name: Handicap
 - Alias: (empty)
 - Type: int
 - Scope: Private
 - Stereotype: (empty)
 - Containment: Not Specified
 - Initial: 40
 - Notes: (empty text area with rich text editor toolbar)
- Attributes Section:**
 - Buttons: New, Copy, Save, Delete
 - Table:

Name	Type	Initial Value
Handicap	int	40
myTag	Color	Yellow
- Bottom Buttons:** Close, Cancel, Help

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 [Add Enumerations to UML Profiles](#)^[14] 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.

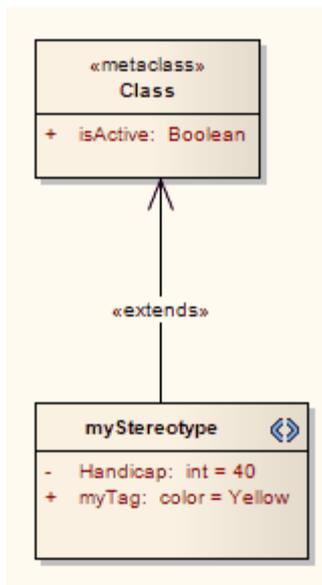
1.2.3.1 With Predefined Tag Types

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](#)^[83] 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](#)^[19] 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 details:

- General Tab:**
 - Name:
 - Alias:
 - Type: ...
 - Scope: ...
 - Stereotype: ...
 - Containment: ...
 - Initial: ...
 - Notes:

B I U A | :≡ ≡≡ | x² x₂ 🌐
- Attributes Section:**
 - Buttons: Attributes, New Copy Save Delete
 - Table:

Name	Type	Initial Value
_image	int	<Image type="EAShapeS...
_sizeX	int	100
- Bottom Buttons:** Close Cancel Help

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

Note:

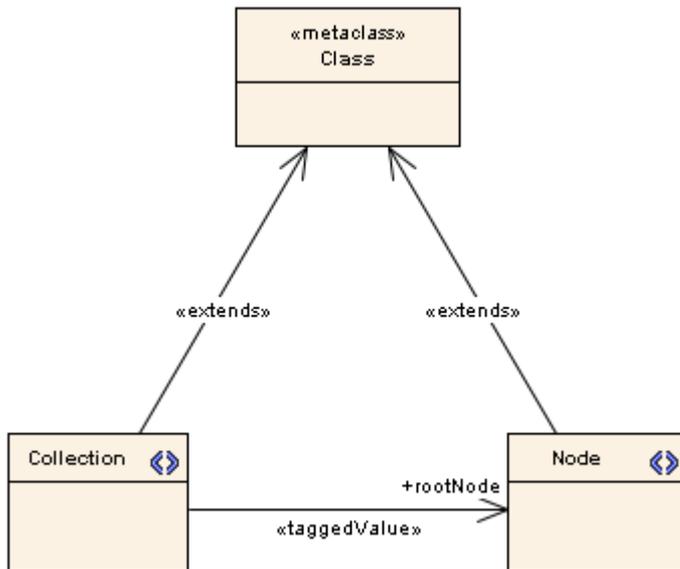
For supported attributes you set only the **Name** (which must match the attributes listed in the supported attributes section) and the **Initial** value; do not set the other values.

4. 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 with the target role 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, through which you locate the 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.

General Details Requirements Constraints Links Scenarios Files Tagged Values

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

B I U A | | | x² x₂ |

Defined Constraints New Save Delete

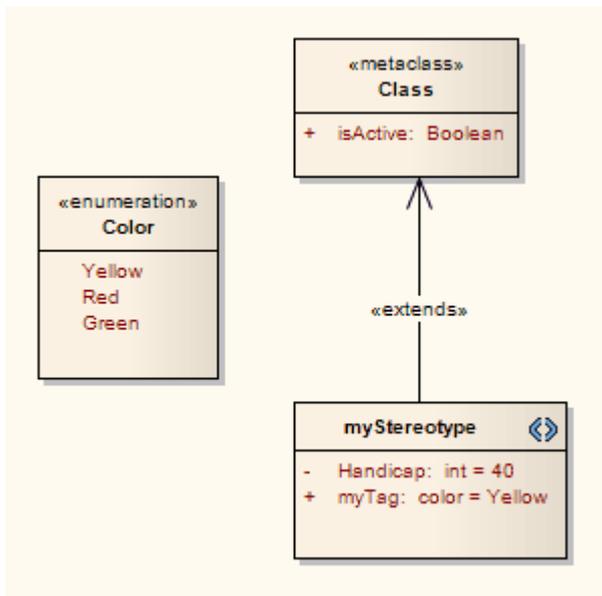
Constraint	Type	Status
t.isRegistered=false	Pre-condition	Approved

OK Cancel Apply Help

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.

General | Detail | Constraints | Tagged Values

Name: Green|

Alias:

Type: int

Scope: Public

Stereotype: enum

Containment: Not Specified

Initial:

Notes:
 B I U \int A | $\frac{1}{3}$ $\frac{1}{3}$ | x^2 x_2

Attributes New Copy Save Delete

Name	Type	Initial Value
Green	int	

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`.

General | Detail | Constraints | Tagged Values

Name:

Alias:

Type: ...

Scope: ...

Stereotype: ...

Containment: ...

Initial: ...

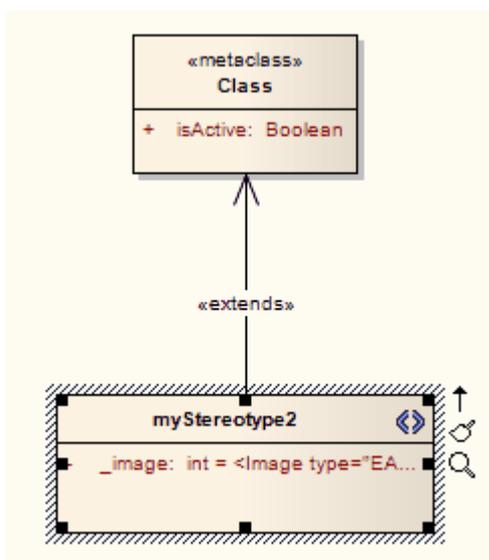
Notes:

B *I* U ^A | ≡ ≡ | x^2 x_2

Attributes

Name	Type	Initial Value
_image	int	

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:



Note:

If you are creating a Shape Script for an Association Class (see *The UML Dictionary*), be aware that the Shape Script is applied to both the Class part and the Association part. Therefore, you might have to include logic in the *shape main* that tests the type of the element so that you can give separate drawing instructions for Class and for Association. Such logic is not necessary in the:

- *shape source* or *shape target*, which are ignored by Classes, or the
- *decoration* shapes which are ignored by Associations.

1.2.7 Set Default Appearance

You can define the appearance of stereotyped elements and connectors as you [create or edit the stereotypes](#) ^[13], 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](#) ^[17] 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 [Save Profile Options](#) ^[18].

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

3. Click on the [...] (Browse) button, and select the export destination for the XML Profile file. If necessary, edit the profile filename, but do not delete the .xml extension.
4. In the **Profile Type** field, click on the drop-down arrow and select **XMI (UML)2.0**.

Note:

The drop-down list is not available unless the package has the `<<profile>>` stereotype.

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** ¹⁷⁷ - 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:

Save From Diagram	Save From Package	Notes
		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 <code>_sizeX</code> , <code>_sizeY</code> and <code>_image</code> properties, but there is no equivalent for default colors.	
Can be much faster.	Can be much slower.	The difference arises because diagram objects are kept in memory and Project Browser 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
icon	Contains the path to a bitmap file to be used as the Project Browser icon for all elements other than <i>Package</i> , with the given stereotype. The bitmap must be 16x16 pixels. For a transparent background, use light grey - RGB(192,192,192).
_image	Shape script definition.
_instanceMode	Used for defining behavior on creating an instance ^[22] .
_instanceOwner	Used for defining behavior on creating an instance ^[22] .
_instanceType	Used for defining behavior on creating an instance ^[22] .
_lineStyle	Sets the line style of a connector. The value of the attribute can be one of: <ul style="list-style-type: none"> • direct • auto • custom • bezier • treeH (horizontal) • treeV (vertical) • treeLH (lateral horizontal) • treeLV (lateral vertical).
_metatype	Used for defining stereotypes as metatypes ^[21] .

Attribute	Meaning
<code>_sizeY</code>	Initial height of the element, in pixels at 100% zoom.
<code>_sizeX</code>	Initial width of the element, in pixels at 100% zoom.
<code>_strictness</code>	Used for restricting application of multiple stereotypes ^[21] .

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:

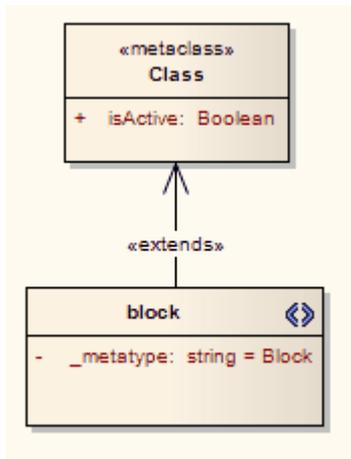
Attribute	Meaning
<code>_AttInh</code>	If set to 1 , switches on the <i>Inherited Features: Show Attributes</i> setting.
<code>_AttPkg</code>	If set to 1 , switches on the <i>Attribute Visibility: Package</i> setting.
<code>_AttPri</code>	If set to 1 , switches on the <i>Attribute Visibility: Private</i> setting.
<code>_AttPro</code>	If set to 1 , switches on the <i>Attribute Visibility: Protected</i> setting.
<code>_AttPub</code>	If set to 1 , switches on the <i>Attribute Visibility: Public</i> setting.
<code>_ConInh</code>	If set to 1 , switches on the <i>Show Element Compartments: Inherited Constraints</i> setting.
<code>_Constraint</code>	If set to 1 , switches on the <i>Show Element Compartments: Constraints</i> setting.
<code>_DefaultDiagramType</code>	Used for defining child diagram types ^[23] .
<code>_HideType</code>	If set to a comma-separated list of stereotypes, sets the <i>Hide Stereotyped Features</i> filter.
<code>_MakeComposite</code>	Used for creating composite elements ^[23] .
<code>_OpInh</code>	If set to 1 , switches on the <i>Inherited Features: Show Operations</i> setting.
<code>_OpPkg</code>	If set to 1 , switches on the <i>Operation Visibility: Package</i> setting.
<code>_OpPri</code>	If set to 1 , switches on the <i>Operation Visibility: Private</i> setting.
<code>_OpPro</code>	If set to 1 , switches on the <i>Operation Visibility: Protected</i> setting.
<code>_OpPub</code>	If set to 1 , switches on the <i>Operation Visibility: Public</i> setting.
<code>_PType</code>	If set to 1 , switches on the <i>Show element type (Port and Part only)</i> setting.
<code>_ResInh</code>	If set to 1 , switches on the <i>Show Element Compartments: Inherited Responsibilities</i> setting.
<code>_Responsibility</code>	If set to 1 , switches on the <i>Show Element Compartments: Responsibilities</i> setting.
<code>_Runstate</code>	If set to 1 , switches on the <i>Hide Object Runstate in current diagram</i> setting.
<code>_SourceAggregation</code>	Used to set the aggregation type at the end of a connector; do not set <i>both</i> <code>_SourceAggregation</code> and <code>_TargetAggregation</code> . Set to 1 for shared, 2 for composite.
<code>_SourceMultiplicity</code>	Used to set the multiplicity of the source element, such as 1..* or 0..1 .
<code>_SourceNavigability</code>	If the connector is non-navigable, set this attribute to Non-Navigable . For other values, set the direction ^[25] attribute.
<code>_Tag</code>	If set to 1 , switches on the <i>Show Element Compartments: Tags</i> setting.
<code>_TagInh</code>	If set to 1 , switches on the <i>Show Element Compartments: Inherited Tags</i> setting.
<code>_TargetAggregation</code>	Used to set the aggregation type at the end of a connector; do not set <i>both</i> <code>_SourceAggregation</code> and <code>_TargetAggregation</code> .

Attribute	Meaning
	Set to 1 for shared, 2 for composite.
_TargetMultiplicity	Used to set the multiplicity of the target element, such as 1..* or 0..1 .
_TargetNavigability	If the connector is non-navigable, set this attribute to Non-Navigable . For other values, set the direction ^[257] attribute.

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:

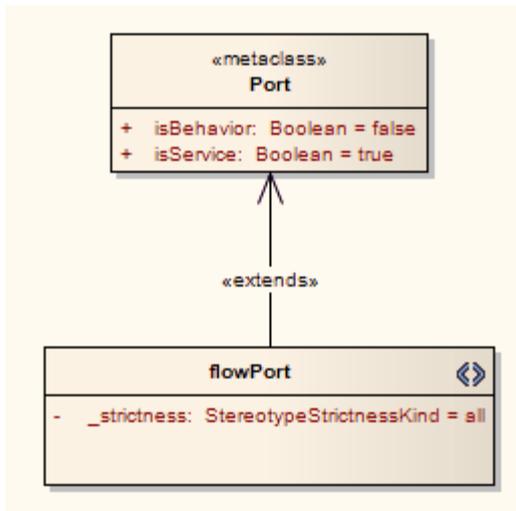
- Shows *Block Properties* rather than *Class Properties* as the title of its **Properties** dialog
- Is auto-numbered as *Block1* not *Class1* on creation, and
- Appears as *Block* not *Class* in many other contexts throughout Enterprise Architect.

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.



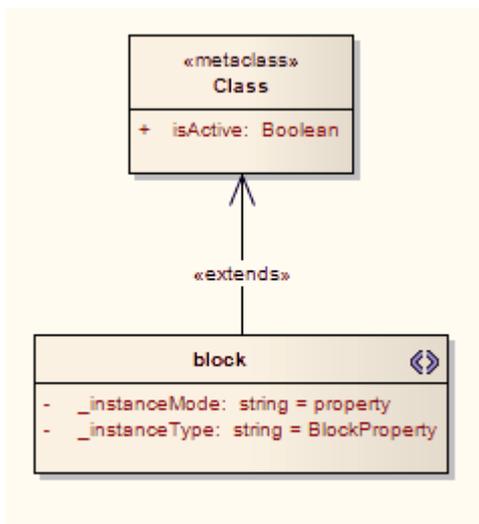
1.2.9.3 Define Creation of Instance

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.



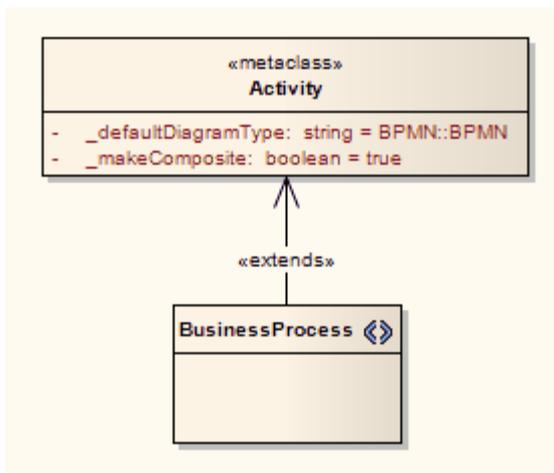
1.2.9.4 Create Composite Elements

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](#)^[23], the child diagram created is a Package diagram.

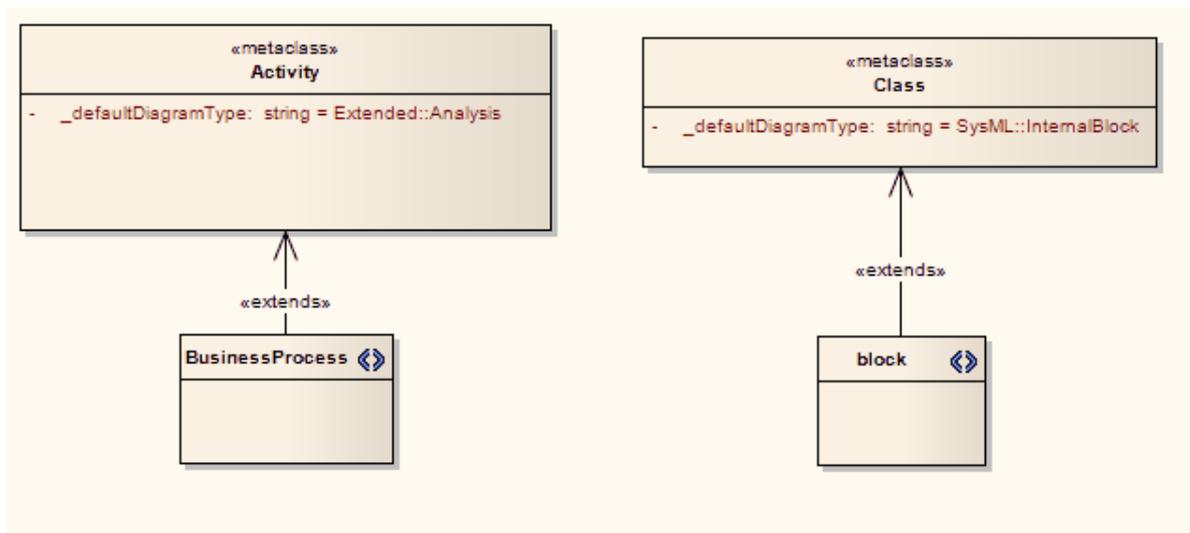
The following example from BPMN shows that a *BusinessProcess* element is always created as a Composite element with a BPMN custom child diagram.



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](#)^[23]. 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 Stereotype 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.

Releases from Enterprise Architect 7 take 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 (e.g. *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](#) ^[25]
- [Quick Linker Example](#) ^[27]
- [Hide Default Quick Linker Settings](#) ^[28]
- [Quick Linker Object Names](#) ^[29].

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 Target Element Type 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 Create Element also set, results in the creation of an element of this type.
G	New Element Stereotype	If set and Create Element also set, results in the creation of an element with this stereotype.
H	New Link Type	If set and Create Link also set, results in the creation of a connector of this type.
I	New Link Stereotype	If set and Create Link also set, results in the creation of a connector with this stereotype.
J	New Link Direction	Can be: <ul style="list-style-type: none"> • directed (always creates an association from source to target) • from (always creates an association from target to source) • undirected (always creates an association with unspecified direction) • bidirectional (always creates a bi-directional association), or • to (creates either a directed or undirected association, depending on the value of the Association Direction option). <p>Note: 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 TRUE , results in creation of a new connector; otherwise should be left blank.
N	Create Element	If set to TRUE 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 Target Element Type and Target Stereotype Filter .
O	Disallow Self connector	Should be set to TRUE 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 TRUE , 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 TRUE 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 TRUE the element being created is embedded in the target element; otherwise should be left blank.
U	Precedes Separator LEAF	If set to TRUE results in a menu separator being added to the Quick Linker menu; otherwise should be left blank.
V	Precedes Separator GROUP	If set to TRUE 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](#) 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:

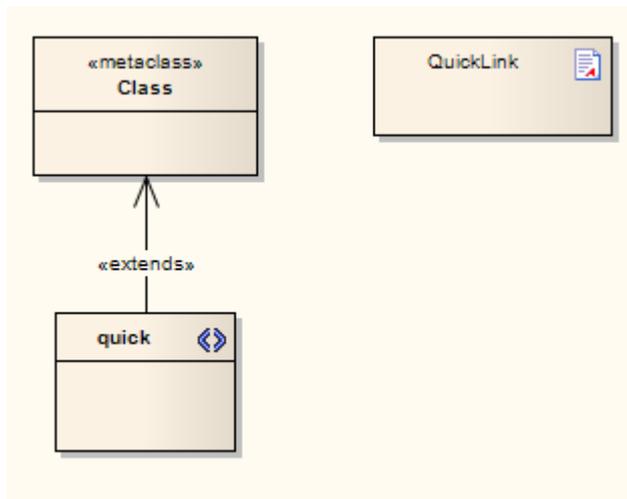
	A	B	C	E	F	H	J	K
1	//Source Element Type	Source ST filter	Target Element Type	Diagram Filter	New Element Type	New Link Type	New Link Direction	New Link Caption
2	Class	quick			Component	Dependency	to	
3	Class	quick			Component	Dependency	from	
4	Class	quick	Component			Dependency	to	Dependency to
5	Class	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](#) ²⁷:

```
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	Package
ActionPin	ExitPoint	Part
Activity	ExitState	Port
ActivityParameter	ExpansionNode	PrimitiveType
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
DataType	InterruptableActivityRegion	StateMachine
Decision	JunctionState	Synchronization_H
DeepHistoryState	MergeNode	Synchronization_V
Deployment Specification	MessageEndpoint	SynchState
Device	n-ary Association	UMLDiagram
DiagramGate	Node	UseCase
EntryPoint	Object	ValueLifeline
EntryState	ObjectNode	

List of Connector Types

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

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

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:

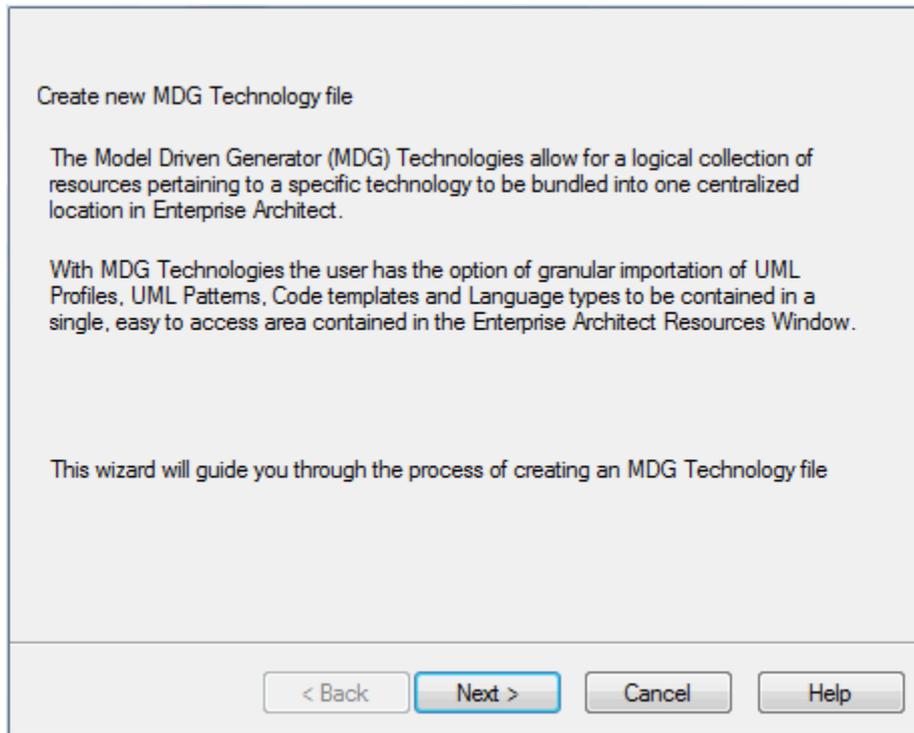
- [Create MDG Technologies](#) ^[30]
- [Working With MTS Files](#) ^[46]
- [Customize Toolbox Profiles](#) ^[47]
- [Create Diagram Profiles](#) ^[52]
- [Create Tasks Pane Profiles](#) ^[54]
- [Define Validation Configuration](#) ^[58]
- [Incorporate Model Templates](#) ^[59]
- [Deploy an MDG Technology](#) ^[59]

An example of creating an MDG Technology for an Enterprise Architecture framework is provided in the white paper: [Enterprise Architecture Framework Design with Sparx Systems Enterprise Architect](#).

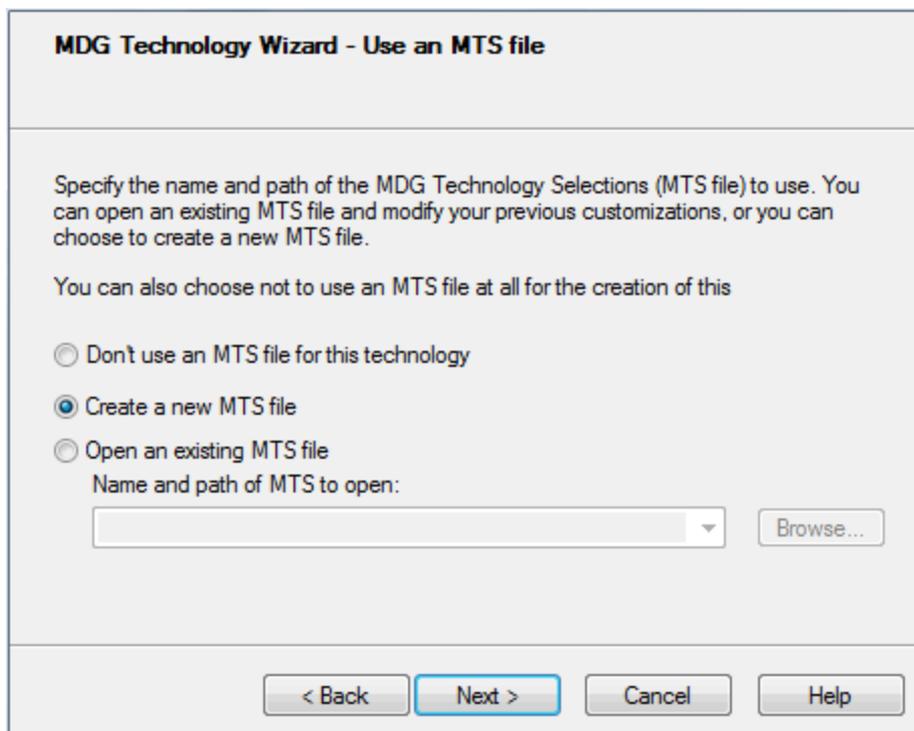
2.1 Create MDG Technologies

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 the 'MDG Technology Wizard - Save MTS file' dialog box. It contains the following elements:

- Title:** MDG Technology Wizard - Save MTS file
- Text:** You can save your changes in the MTS file that you opened, or enter the name and path of a new MTS file.
- Input Field:** Name and path of MTS to save: Y:\Dev\Roy\MY_MTS.mts
- Button:** Browse...
- Navigation Buttons:** < Back, Next >, Cancel, 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 the 'MDG Technology Wizard - Create' dialog box. It contains the following elements:

- Title:** MDG Technology Wizard - Create
- Text:** Please specify the technology to be created
- Fields:**
 - Technology: MDG Technology For Workflows
 - Filename: %oy\UML Profiles\WKFL\MDG Technology For Workflows.xml
 - ID: WKFL
 - Version: 0.1
 - Icon: Y:\Dev\Help\Source\EA Next\Help_Graphics\uml_profiles.bn
 - Logo: Y:\Dev\Help\Source\EA Next\Help_Graphics\Banner1.BMP
 - URL: (empty)
 - Support: (empty)
 - Notes: Technology to generate and collate project development workflows.
- Navigation Buttons:** < Back, Next >, Cancel, Help

5. Complete the fields on this screen as follows:

Option	Use to
Technology	Type the name of the MDG Technology.
Filename	Type or select the path and filename of the MDG Technology File (the file extension for this file is .xml).
ID	Type a reference for the MDG Technology File, up to 12 characters long.
Version	Type the version number of the MDG Technology File.
Icon	(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 MDG Technologies dialog.
Logo	(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 MDG Technologies dialog.
URL	(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.
Support	(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.
Notes	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

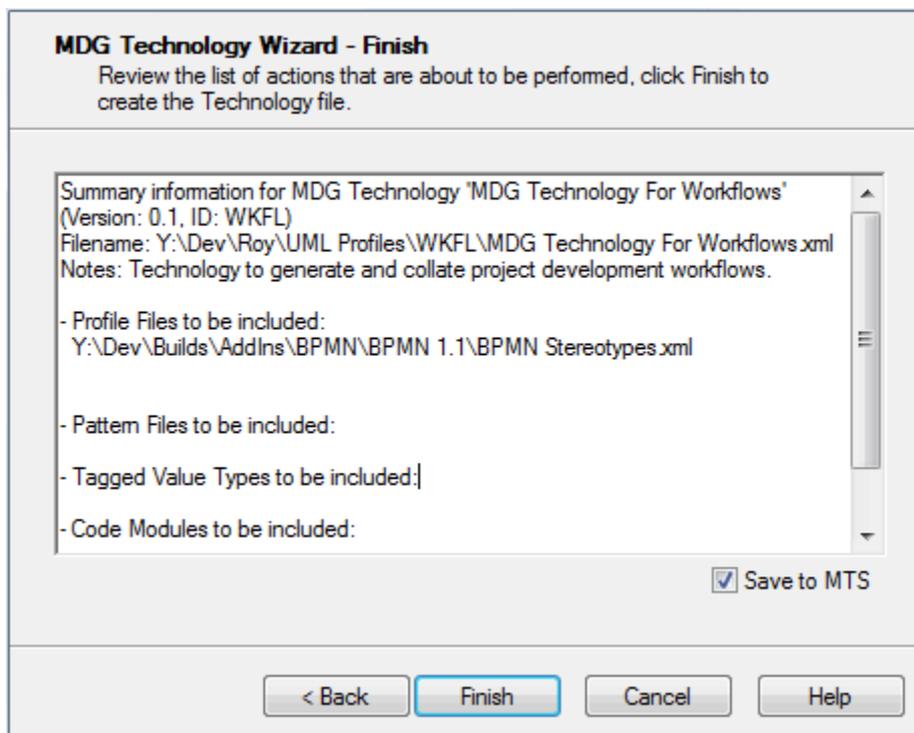
- Images
- Scripts

< 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](#) ^[35]
 - [Add a Pattern](#) ^[35]
 - [Add a Diagram Profile](#) ^[36]
 - [Add a Toolbox Profile](#) ^[37]
 - [Add Task Panel Pages](#) ^[38]
 - [Add Tagged Value Types](#) ^[39]
 - [Add Code Modules](#) ^[40]
 - [Add MDA Transformations](#) ^[42]
 - [RTF Report Templates](#) ^[44]
 - [Linked Document Templates](#) ^[45]
 - [Add Images](#) ^[43]
 - [Add Scripts](#) ^[43] (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](#) ^[46], if required, to add further items such as:

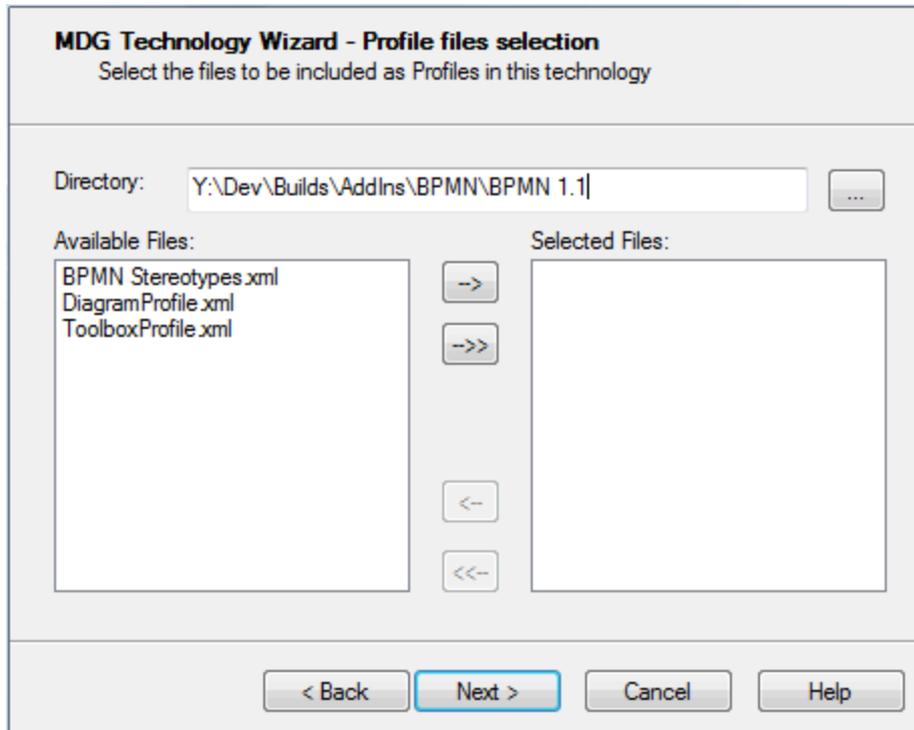
- Model Search definitions
- Model Views
- Model Validation configurations
- Model Templates.

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](#)^[5]. To use the Profiles section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)^[30] topic up to and including [Step 6](#)^[33], 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:

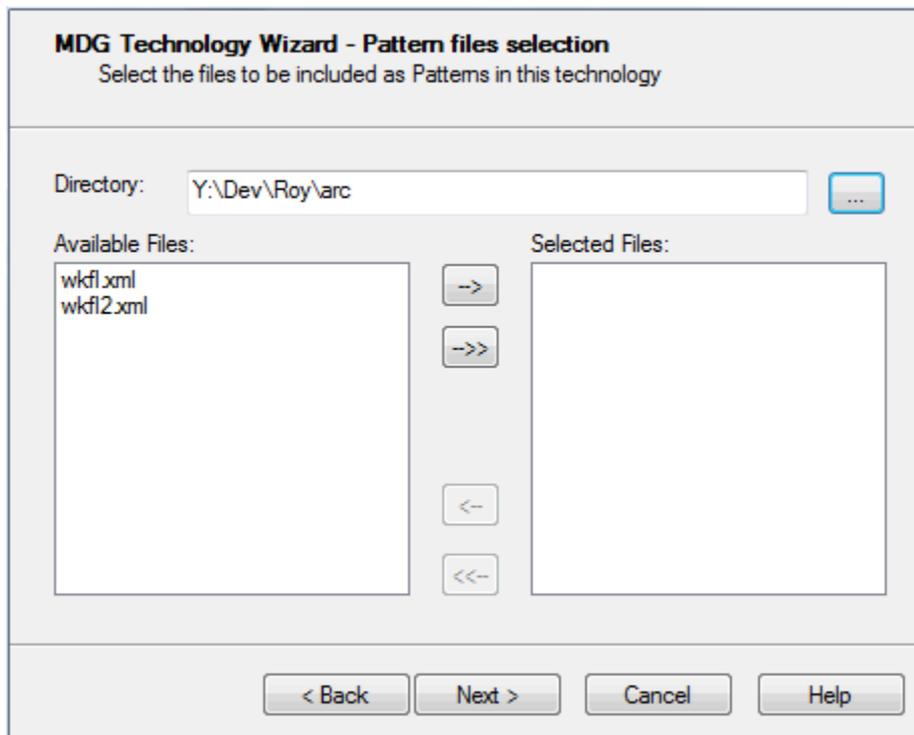
- DO NOT select diagram profiles or toolbox profiles on this dialog; this would generate conflicting commands in the .MTS file.
- Make sure you do include your [stereotypes profile](#)^[24].

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

2.1.2 Add a Pattern

When creating an MDG Technology file, you can include patterns (see *Extending UML With Enterprise Architect*). To use the *Patterns* section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Patterns** checkbox. The [MDG Technology Wizard - Pattern files selection](#) dialog displays.

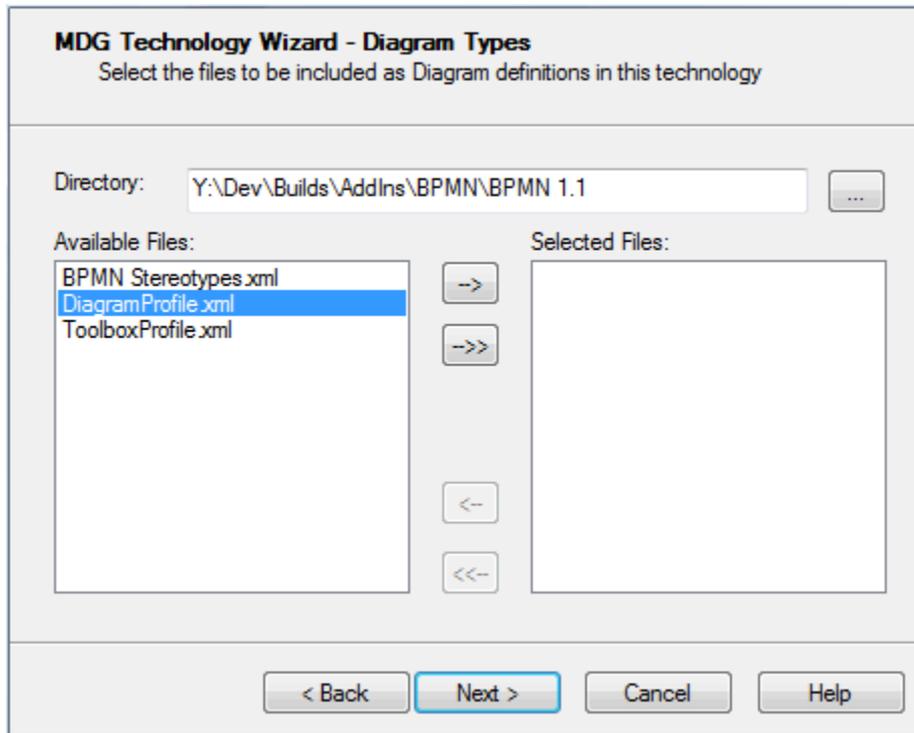


2. In the **Directory** field, navigate to the directory containing the required pattern or patterns. The pattern files are automatically listed in the **Available Files** panel.
3. To select each required pattern individually, highlight the pattern in the **Available Files** list and click on the --> 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](#)^[52]. To use the diagram profiles section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Diagram Types** checkbox. The **MDG Technology Wizard - Diagram Types** dialog displays.

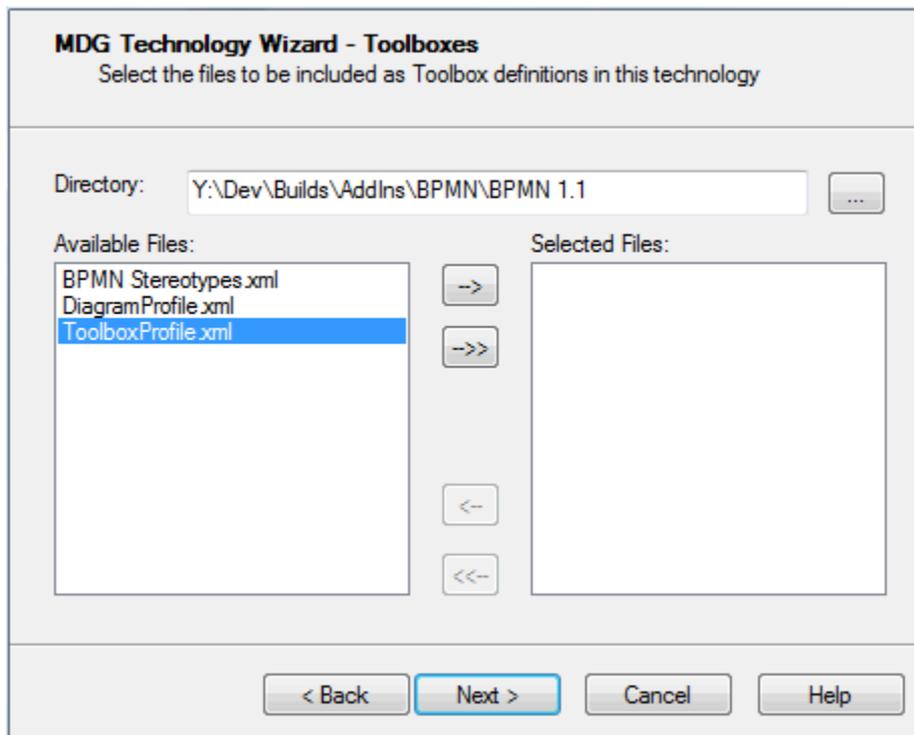


2. In the **Directory** field, navigate to the directory containing the required diagram profiles. The profiles in the directory are automatically listed in the **Available Files** panel.
3. To select each required diagram profile individually, highlight the file name in the **Available Files** list 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](#)^[47]. To use the Toolboxes section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Toolboxes** checkbox. The **MDG Technology Wizard - Toolboxes** dialog displays.

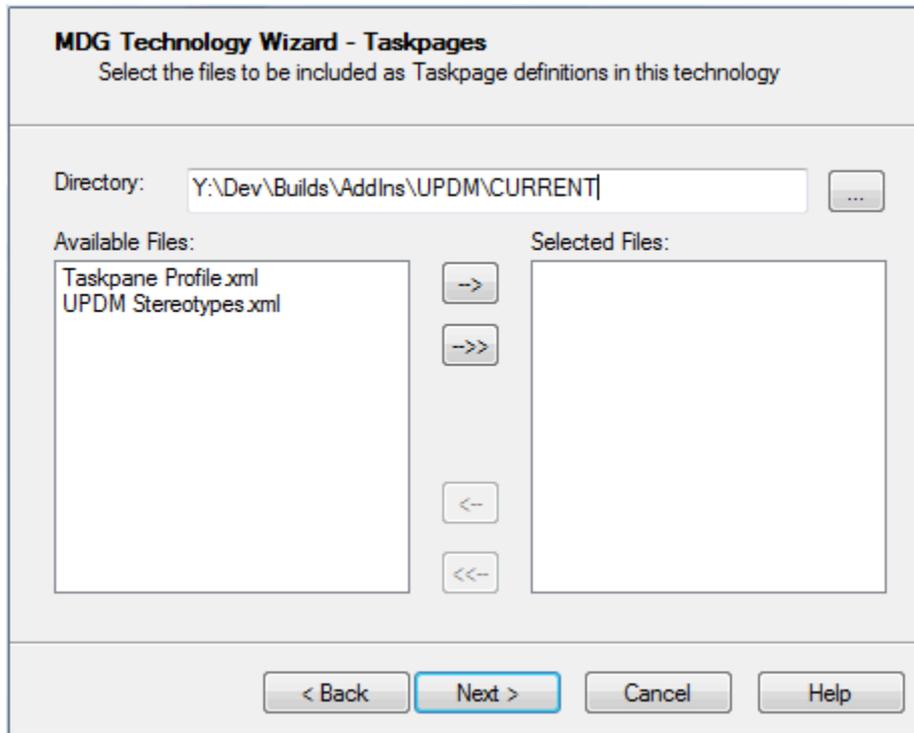


2. In the **Directory** field, navigate to the directory containing the required toolbox profiles. The profile files are automatically listed in the **Available Files** panel.
3. To select each required toolbox profile individually, highlight the file name in the **Available Files** list and click on 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](#)^[54]. To use the Taskpages section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Taskpages** checkbox. The **MDG Technology Wizard - Taskpages** dialog displays.

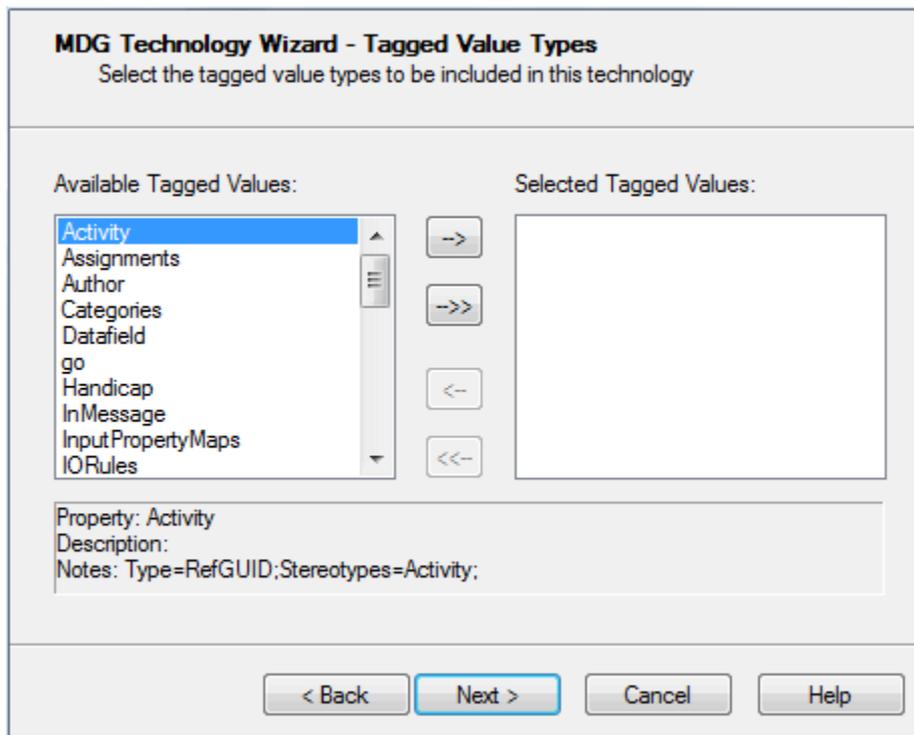


2. In the **Directory** field, navigate to the directory containing the required taskpage profiles. The profile files are automatically listed in the **Available Files** panel.
3. To select each required taskpage profile individually, highlight the file name in the **Available Files** list and click 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](#)^[81]. 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](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Tagged Value Types** checkbox. The **MDG Technology Wizard - Tagged Value Types** dialog displays.

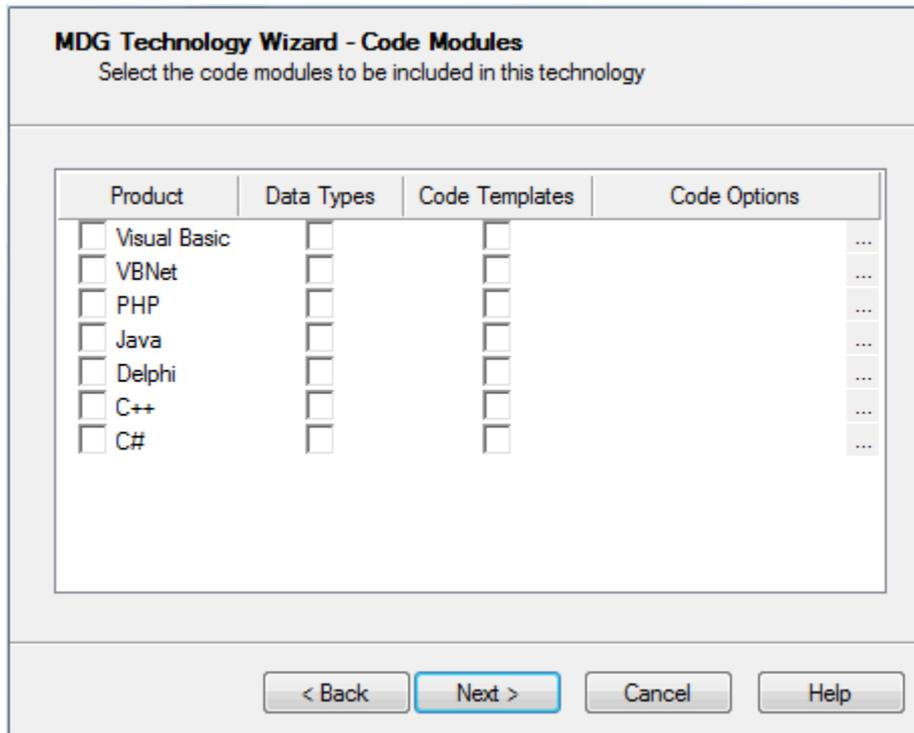


2. To select each required Tagged Value Type individually, highlight the file name in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available Tagged Value Types, click on the -->> button.
3. Click on the **Next** 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](#)^[30] topic up to and including [Step 6](#)^[33], 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 classHasConstructor code template macro.
CopyConstructorName	The name of a function used as a copy constructor. Used by the classHasCopyConstructor 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 classHasDestructor code template macro.
Editor	The external editor used for editing source of this language.
ImplementationExtensio	The extension used by Enterprise Architect to generate an implementation

Code Option	Description
n	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>
```

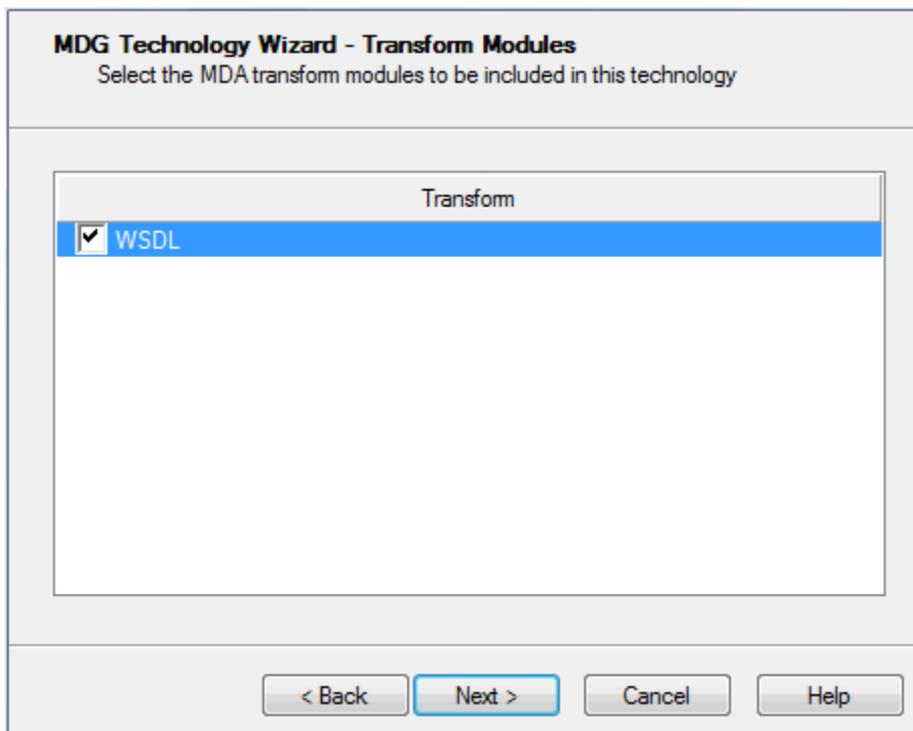
4. Click on the **Next** button to proceed.

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](#) ³⁰ topic up to and including [Step 6](#) ³³, where you select the **MDA Transforms** checkbox. The [MDG Technology Wizard - Transform Modules](#) dialog displays.

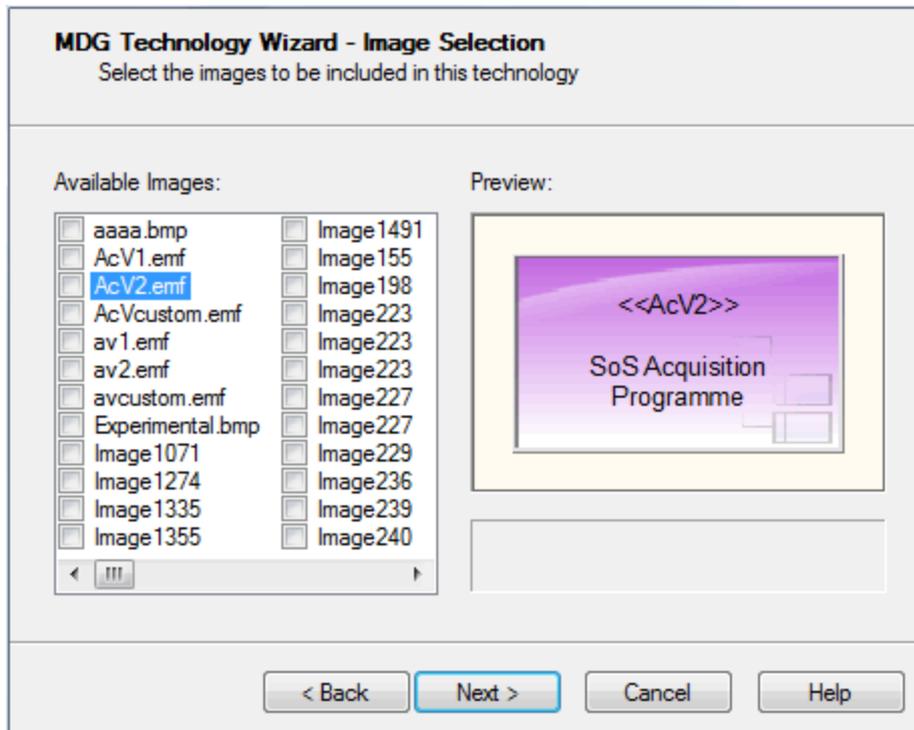


2. Click the checkbox against the template name of each required template that is present in the current model.
3. Click on the **Next** button to proceed.

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](#)^[30] topic up to and including [Step 6](#)^[33], 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

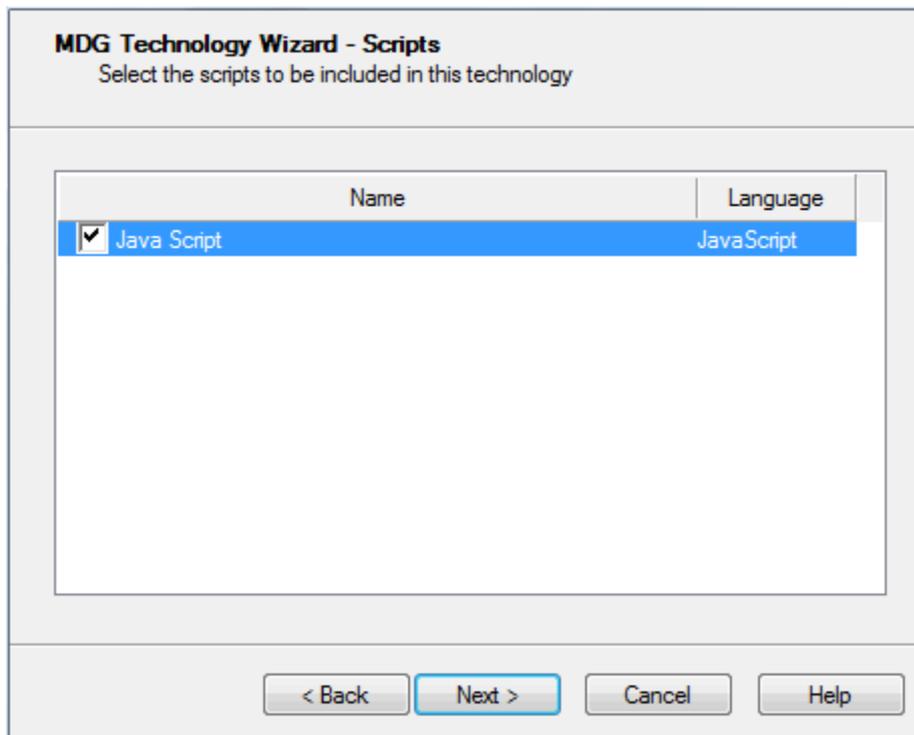
When creating an MDG Technology file, you can include scripts that you have created in the model (see *Using Enterprise Architect - UML Modeling Tool*).

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](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **Scripts** checkbox. The **MDG Technology Wizard - Scripts** dialog displays.

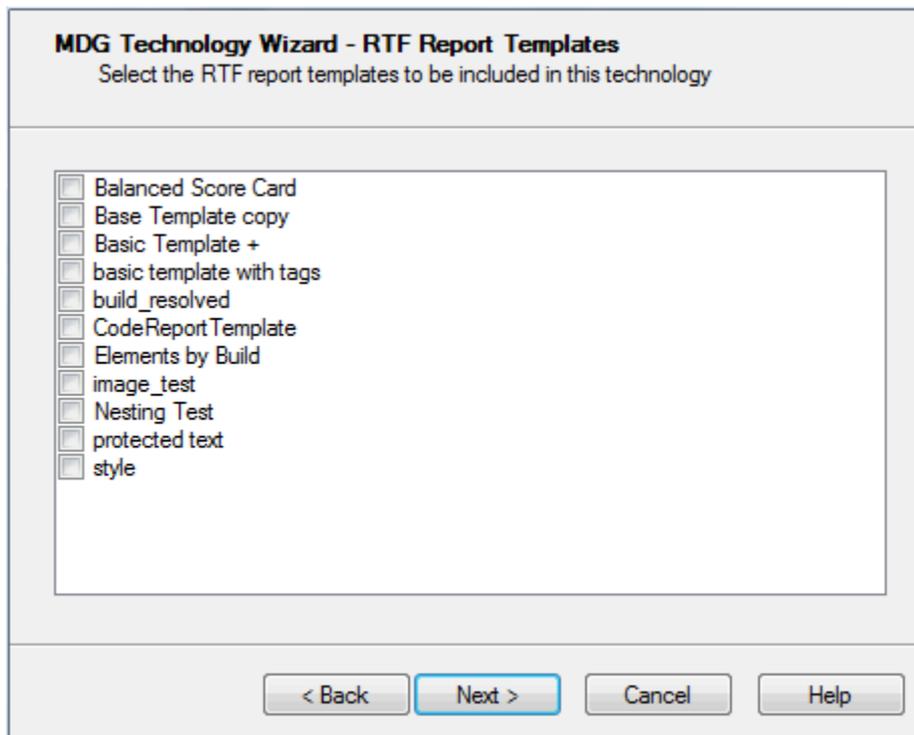


2. For each required script available in the current model, select the checkbox next to the script name.
3. Click on the **Next** button to proceed.

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](#)^[30] topic up to and including [Step 6](#)^[33], where you select the **RTF Templates** checkbox. The **MDG Technology Wizard - RTF Report Templates** dialog displays.

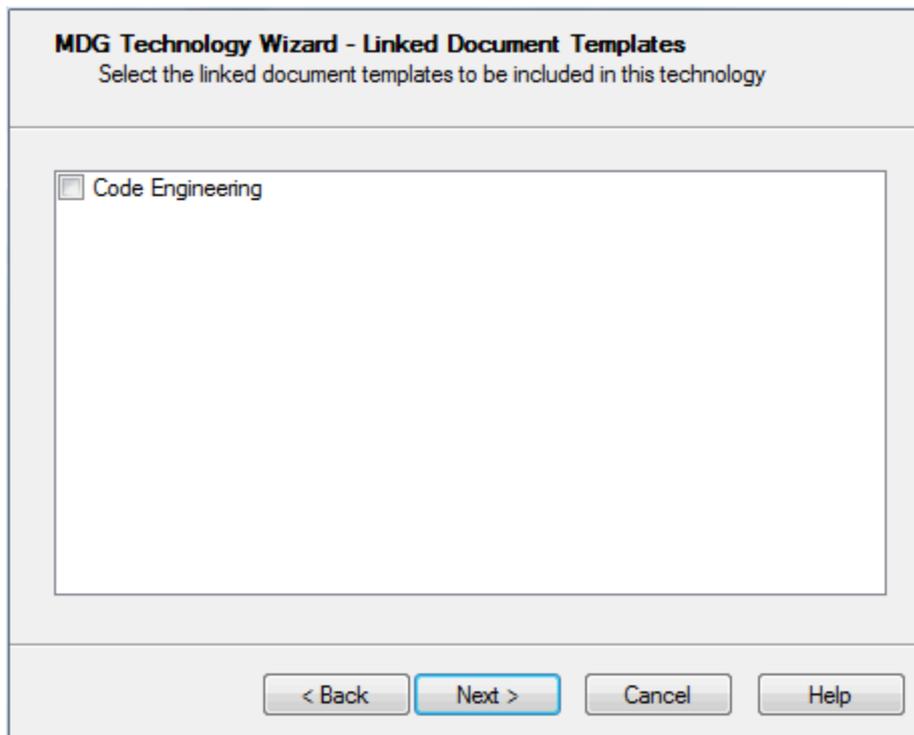


2. For each required user-defined report template available in the current model, select the checkbox next to the template name.
3. Click on the **Next** button to proceed.

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](#)^[30] topic up to and including [Step 6](#)^[33], 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](#)^[30]. 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](#)^[58]
- [Model Templates](#)^[59]

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](#)^[47] that contain the definitions that Enterprise Architect requires to create the toolboxes.
2. [Create a .MTS file](#)^[30] 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](#)^[48]
 - [Override Enterprise Architect's default toolboxes](#)^[49]
 - Change the default [icons for toolbox items](#)^[49].

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](#)^[48].
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](#)^[50] and [connectors](#)^[51] 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 ID of the technology and *MyPattern* is the name of the pattern to drop. For example, *BusFramework::Builder(UMLPattern)*.

To define a model-based pattern in a custom toolbox (such as the GoF patterns - see *Extending UML With Enterprise Architect*) create an attribute with a name of the format *PatternCategory::PatternName(UML Pattern)*. For example: *GoF Behavioral Patterns:: Mediator(UML Pattern)*.

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*.

- 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

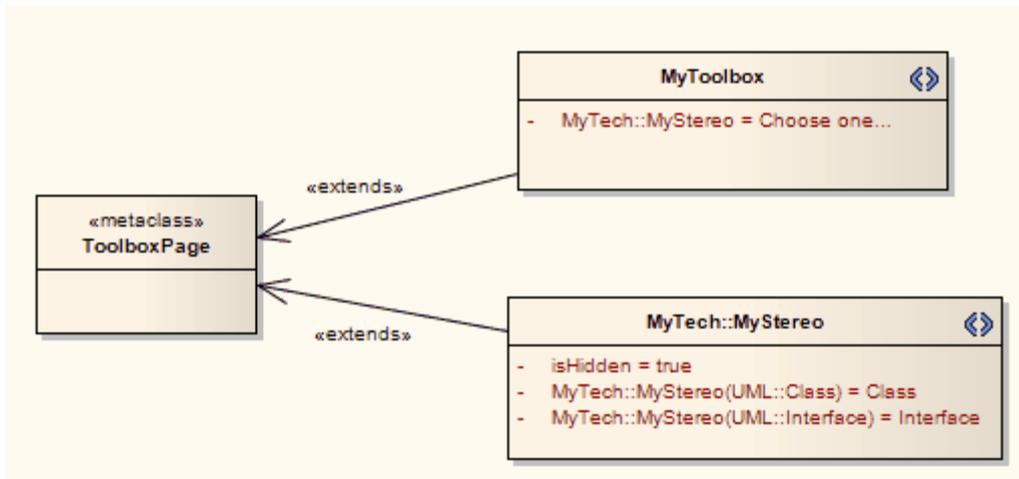
The following attributes can be added to a stereotype Class that extends the *ToolboxPage* metaclass:

- **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; the page is initially displayed as collapsed
- **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](#)^[49]
- **isHidden:** see [Create Hidden Sub-Menus](#)^[48].

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 toolbox defined by this profile replaces the Enterprise Architect toolbox `UML::Class` as the default toolbox for all UML Class diagrams. For a list of inbuilt toolboxes, see the [Enterprise Architect Toolboxes](#) ⁵⁰ topic.

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](#) ⁵¹.

- 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](#) ⁵⁰.

- 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**)

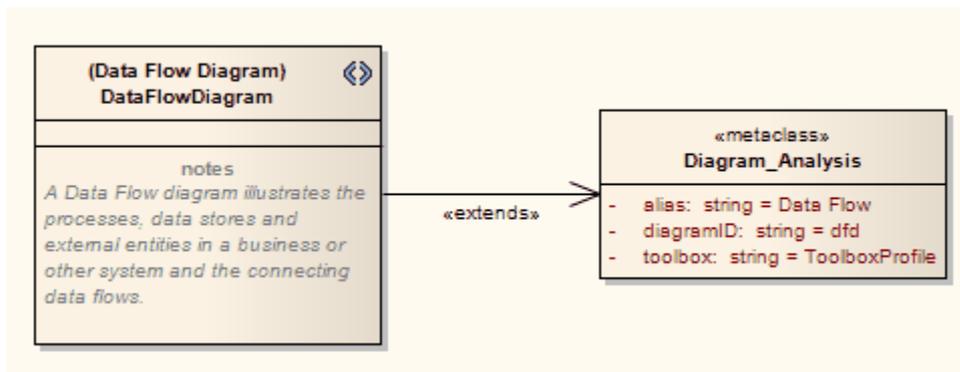
2.4 Create Diagram Profiles

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](#)^[53] 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](#)^[53]
 - *pdata*: *string* = one or more of a range of values; see [Attribute Values - stylex & pdata](#)^[53].
- 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](#)^[36].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](#)^[23], or prefixed by *Diagram_* in creating [Diagram Profiles](#)^[52].

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).

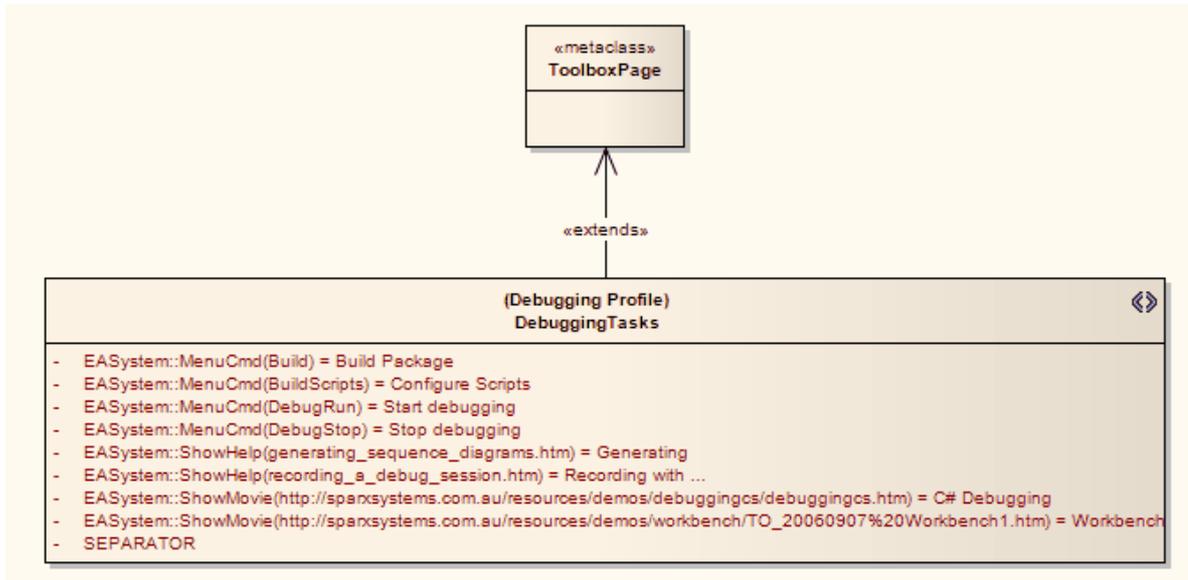
2.5 Create Tasks Pane Profiles

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

1. [Define Toolboxes](#)^[55]. Create any number of stereotype elements that each define a **Tasks Pane** toolbox page
2. [Define Contexts](#)^[57]. 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](#)^[57] to Toolboxes. Define the many-to-many relationships between the **Tasks Pane** toolboxes and the available contexts.
4. [Create the Profile](#)^[58] and [incorporate it](#)^[38] 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](#)^[55]. 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](#)^[56] from the **Tasks Pane**.

Next Step

The next step is to create a set of [Tasks Pane Contexts](#)^[57].

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 (for **Team Review** window)
- ViewHierarchy (for **Traceability** window)
- 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

```

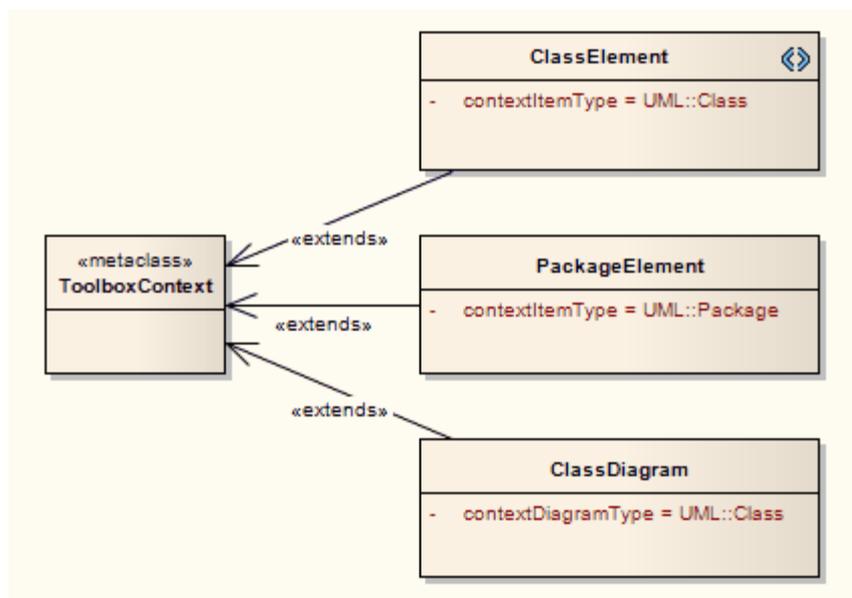
2.5.2 Define Tasks Pane Contexts

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](#) [55]. 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

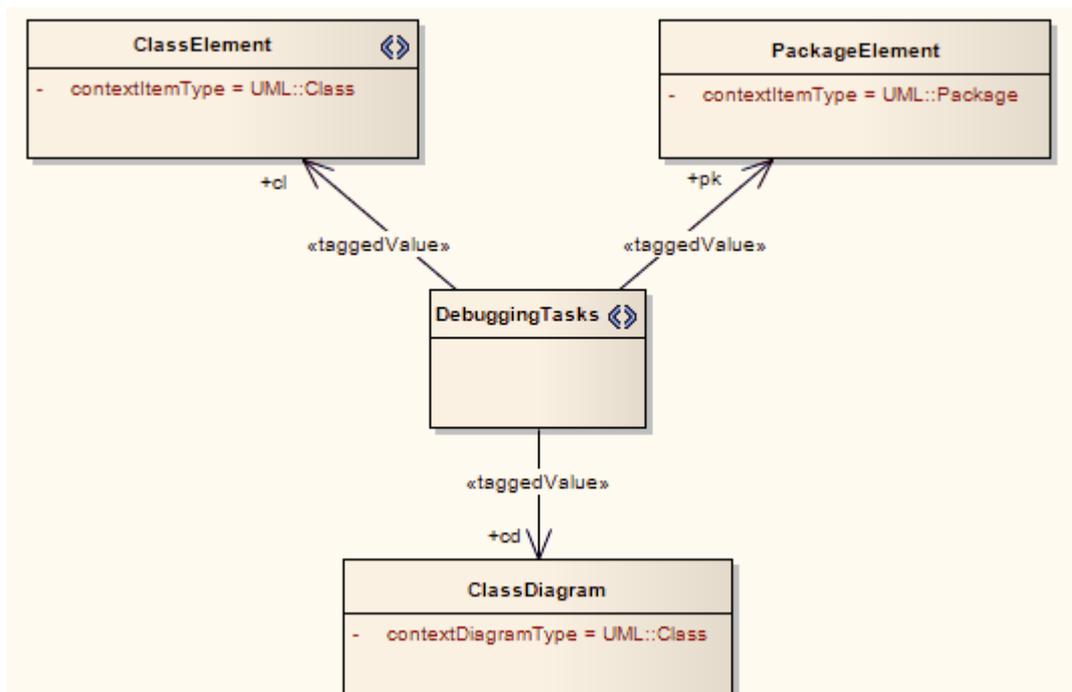


Next Step

The next step is to [allocate Tasks Pane contexts to Tasks Pane toolboxes](#) [57].

2.5.3 Allocate Tasks Pane Contexts

Once you have defined your [Tasks Pane toolboxes](#) [55] and [Tasks Pane contexts](#) [57], 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](#) ⁵⁸.

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](#) [27] 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](#)^[152] 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



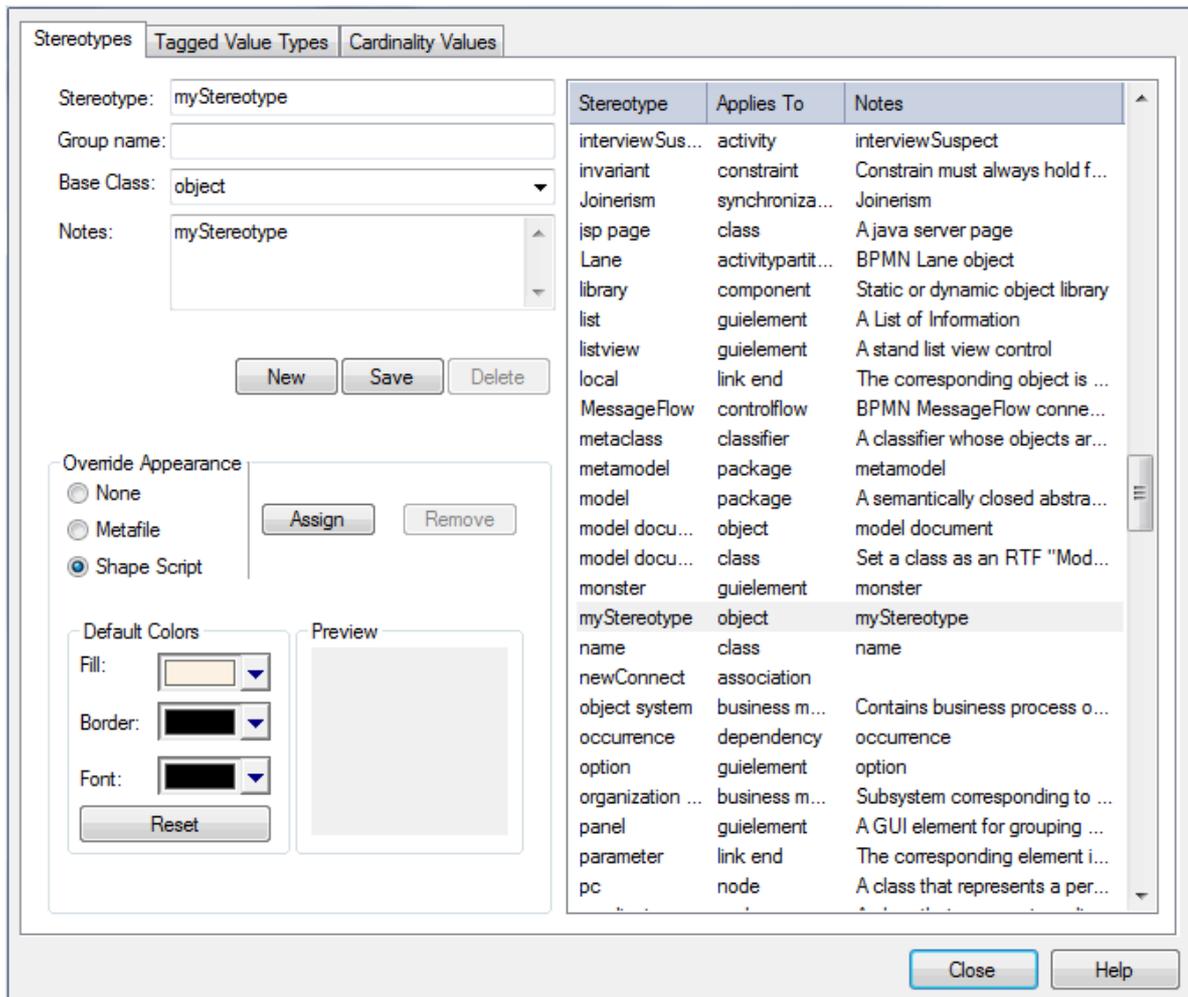
Introduction

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](#) ⁶¹
- [Write Scripts](#) ⁶⁵
- [Example Scripts](#) ⁷⁷
- [Shape Editor](#) ⁶⁴
- [Add Shape Scripts to UML Profiles](#) ¹⁵

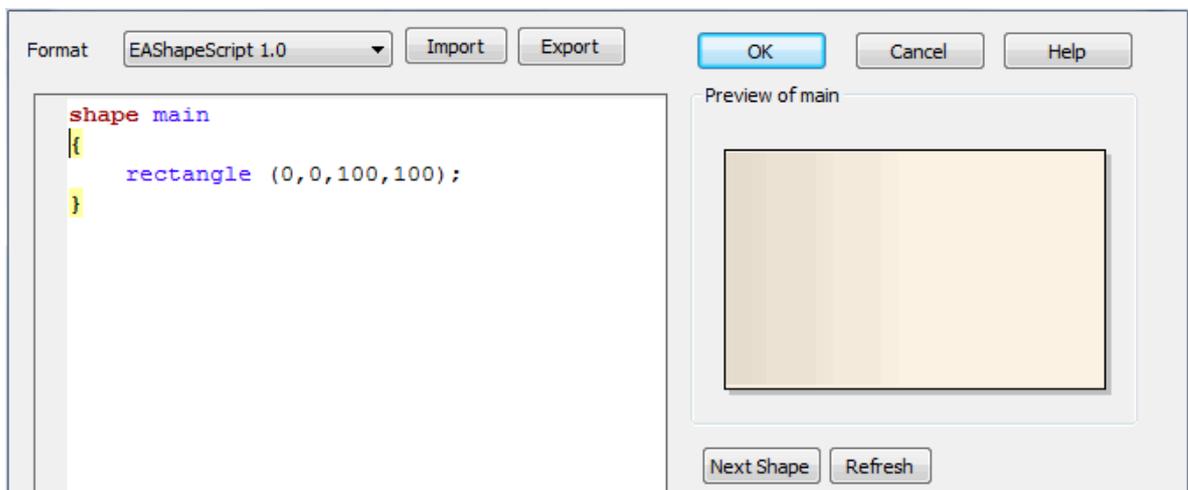
3.1 Getting Started With Shape Scripts

[Shape Scripts](#) ⁶¹ 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](#) ^[64] 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](#)^[77] topic), click on the **Next Shape** button to page through the components of the shape.

Once you have finished [writing your Shape Script](#)^[65], 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.

Note:

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 *UML Modeling With 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.

General Details Requirements Constraints Links Scenarios Files Tagged Values

Name: Class 1

Stereotype: architecture Abstract

Author: Frederick Walter Status: Proposed

Scope: Public Complexity: Easy

Alias: Language: Java

Persistence: Keywords:

Phase: 1.0 Version: 1.0

Notes:

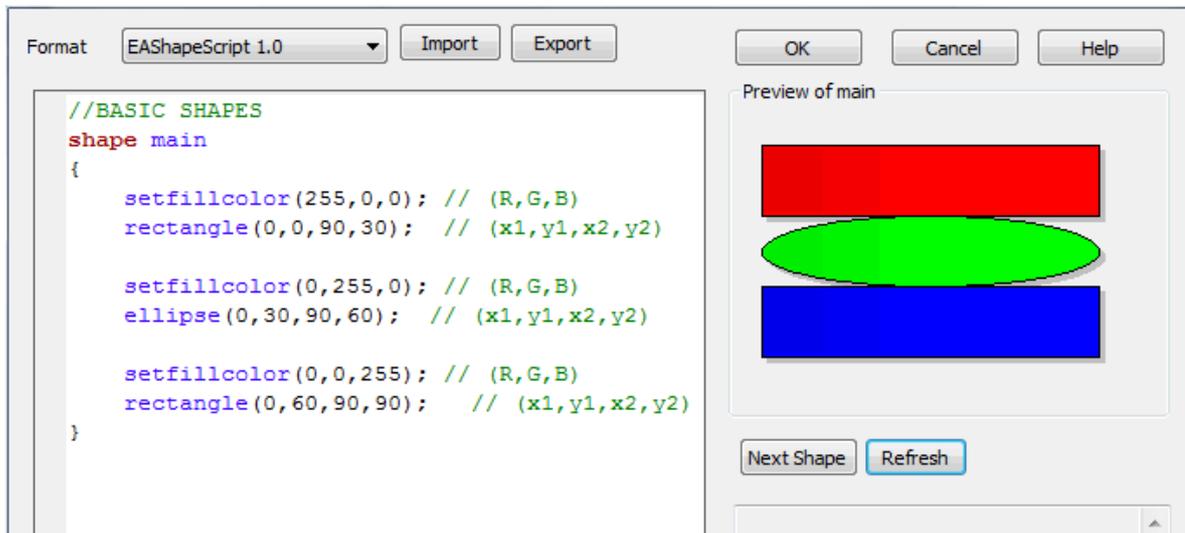
B I U A | ☰ ☷ | x² x₂ 🌐

3.2 Shape Editor

The **Shape Editor** enables you to create [Shape Scripts](#)^[61]. 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
Format	Select the Shape Script version.
Import	Import a Shape Script from a text file.
Export	Export a Shape Script to a text file.
OK	Exit from the Shape Editor , don't forget to save your script from the Stereotypes tab. See Getting Started ^[61] .
Next Shape	Rotate though the multiple shape definitions.
Refresh	Parse your script and display the result in the Preview window.

3.3 Write Scripts

This topic is a detailed reference for writing [Shape Scripts](#) ^[61]. For an introduction to writing Shape Scripts, see the [Getting Started](#) ^[61] and [Example Scripts](#) ^[77] topics.

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

- [Syntax grammar](#) ^[65]
- [Shape attributes](#) ^[66]
- [Drawing methods](#) ^[68]
- [Color queries](#) ^[72]
- [Conditional branching](#) ^[72]
- [Query methods](#) ^[72]
- [Display Item properties](#) ^[72]
- [Sub-shapes](#) ^[75]
- [Reserved names](#) ^[76]
- [Miscellaneous](#) ^[76]

3.3.1 Syntax Grammar

Grammar symbols:

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

- | = or
- ; = terminator

ShapeScript	::=	<Shape>*;
Shape	::=	<ShapeDeclaration> <ShapeBody>;
ShapeDeclaration	::=	<ShapeType> <ShapeName>;
ShapeType	::=	"shape" "decoration";
ShapeName	::=	<ReservedShapeName> <stringliteral>;
ReservedShapeName	::=	See Reserved Names ^[76] for full reserved shape listing
ShapeBody	::=	"{" <InitialisationAttributeAssignment>* <DrawingStatement>* <SubShape>* "}";
InitialisationAttributeAssignment	::=	<Attribute> "=" <Value> ",";
Attribute	::=	See Shape Attributes ^[66] for full listing of attribute names
DrawingStatement	::=	<IfElseSection> <Method>;
IfElseSection	::=	"if" "(" <QueryExpression> ")" <TrueSection> [<ElseSection>];
QueryExpression	::=	<QueryName> "(" <ParameterList> ")";
QueryName	::=	See Query Methods ^[72] for a full listing of Query names
TrueSection	::=	"{" <DrawingStatement>* }"
ElseSection	::=	"else" "{" <DrawingStatement>* }"
Method	::=	<MethodName> "(" <ParameterList> ")" ";"
MethodName	::=	See Drawing Methods ^[68] 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");
}

```

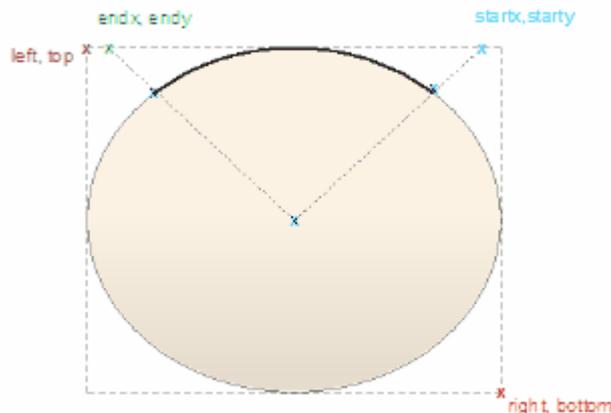
Attribute Name	Type	Description
bottomAnchorOffset	(int,int)	When creating a Shape Script for an embedded element (such as a Port), use this attribute to offset the shape from the bottom edge of its parent. For example: <i>bottomAnchorOffset=(0,-10);</i> move embedded element up 10 pixels from the bottom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

Attribute Name	Type	Description
dockable	<i>string</i>	Makes the shape default to dockable, so that it can be aligned with and joined to other elements (both other Shape Scripts and standard elements) on a diagram. You cannot reverse the dockable status with the Appearance menu option; to change the status, you must edit the Shape Script. Valid values: standard or off
editableField	<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: alias, name, note, stereotype
endPointY, endPointX	<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: 0 and 0
fixedAspectRatio	<i>string</i>	Set to true to fix the aspect ratio. Do not use if you do not want to fix the aspect ratio.
h_Align	<i>string</i>	Affects horizontal placement of printed text and subshapes depending on the layoutType attribute. Valid values: left, center, or right
layoutType	<i>string</i>	Determines how subshapes are sized and positioned. See Subshape Layout ^[75] for further details. Valid values: leftright, topdown, border
leftAnchorOffset	<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
noShadow	<i>string</i>	Set to true to suppress the shapes shadow from being rendered. Valid values: true or false (default= false)
orientation	<i>string</i>	Applies to decoration shapes only. Determines where the decoration is positioned within the containing element glyph. Valid values: NW, N, NE, E, SE, S, SW, W
preferredHeight		Used by border layoutType - north and south Used in drawing the source and target shapes for connectors to determine how wide the line is.
preferredWidth		Used by border layoutType - east and west. Used by leftright layoutType, shapes where <i>scalable</i> is false to determine how much space they occupy for layout purposes.
rightAnchorOffset	<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
rotatable	<i>string</i>	Set to false 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: true or false (default = true)
scalable	<i>string</i>	Set to false to stop the shape from being relatively sized to the associated diagram glyph. Valid values: true or false (default= true)
topAnchorOffset	<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
v_Align	<i>string</i>	Affects vertical placement of printed text and subshapes depending on the layoutType attribute. Valid values: top , center , or bottom

3.3.3 Drawing Methods

Method Name	Description
addsubshape (string shapename[, int width, int height])	Adds a sub-shape with the name shapename that must be defined within the current shape definition.
arc (int left, int top, int right, int bottom, int startingpointx, int startingpointy, int endingpointx, int endingpointy)	Draws an elliptical anticlockwise arc with the ellipse having extents at left , top , right and bottom . 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 (startingpointx , startingpointy). 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 (endingpointx , endingpointy). For example: Arc(0, 0, 100, 100, 95, 0, 5, 0);
arco (int left, int top, int 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
int bottom, int startingpointx, int startingpointy, int endingpointx, int endingpointy)	
bezierto(int controlpoint1x, int controlpoint1y, int controlpoint2x, int controlpoint2y, int endpointx, int endpointy)	Draws a bezier curve and updates the pen position.
defSize(int width, int height)	<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, [Alt]+[Z] also resizes the shape to the defined dimensions.</p> <p>Note:</p> <p>The minimum value for both int width and int height is 10.</p>
drawnativeshape()	<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>
ellipse(int left, int top, int right, int bottom)	Draws an ellipse with extents defined by left , top , right and bottom .
endpath()	Ends the sequence of drawing commands that define a path.
fillandstrokepath()	Fills the previously defined path with the current fill color, then draws its outline with the current pen.
fillpath()	Fills the previously defined path with the current fill color.
hidelabel(string labelname)	Hides the label specified by labelname .
image(string imageld,	Draws the image that has the name imageld in the Image Manager .

Method Name	Description
int left, int top, int right, int bottom)	<p>Note:</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>
lineto(int x, int y)	<p>Draws a line from the current cursor position to a point specified by x and y, and then updates the pen cursor to that position.</p>
moveto(int x, int y)	<p>Moves the pen cursor to the point specified by x and y.</p>
polygon(int centerx, int centery, int numberofsides, int radius, float rotation)	<p>Draws a regular polygon with center at the point (centerx, centery), and numberofsides number of sides.</p>
print(string text)	<p>Prints the specified text string.</p> <p>Note:</p> <p>You cannot change the font size, type or color of this text.</p>
printifdefined(string propertyname, string truepart[, string falsepart])	<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>Note:</p> <p>You cannot change the font size, type or color of this text.</p>
println(string text)	<p>Appends a line of text to the shape and a line break.</p> <p>Note:</p> <p>You cannot change the font size, type or color of this text.</p>
printwrapped(string text)	<p>Prints the specified text string, wrapped over multiple lines if the text is wider than its containing shape.</p> <p>Note:</p> <p>You cannot change the font size, type or color of this text.</p>
rectangle(int left, int top, int right, int bottom)	<p>Draws a rectangle with extents at left, top, right, bottom. Values are percentages.</p>
roundrect(int left, int top, int right, int bottom, int abs_cornerwidth, int	<p>Draws a rectangle with rounded corners, with extents defined by left, top, right and bottom. The size for the corners is defined by abs_cornerwidth and abs_cornerheight; 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 Color Queries [72].
<code>setfillcolor(</code> <code>int red,</code> <code>int green,</code> <code>int blue)</code> <code>setfillcolor(</code> <code>Color newColor)</code>	<p>Sets the fill color.</p> <p>You can specify the required color by defining RGB values or using a color value returned by any of the Color Queries [72]; for example:</p> <pre>GetUserFillColor()</pre>
<code>setlinestyle(</code> <code>string linestyle)</code>	<p>Changes the stroke pattern for commands that use the pen.</p> <p>Parameters:</p> <p>string linestyle: the following styles are valid:</p> <ul style="list-style-type: none"> • solid • dash • dot • dashdot • dashdotdot
<code>setorigin(</code> <code>string relativeTo,</code> <code>int xOffset,</code> <code>int yOffset)</code>	<p>Positions floating text labels relative to the main shape.</p> <p>relativeTo is one of N, NE, E, SE, S, SW, W, NW, CENTER</p> <p>xOffset and yOffset are in pixels, not percentage values, and can be negative.</p>
<code>setpen(</code> <code>int red,</code> <code>int green,</code> <code>int blue[,</code> <code>int penwidth])</code>	<p>Sets the pen to the defined color and optionally sets the pen width.</p> <p>Note:</p> <p>This method is only for line-drawing commands. It does not affect any text commands.</p>
<code>setpencolor(</code> <code>int red,</code> <code>int green,</code> <code>int blue)</code> <code>setpencolor(</code> <code>Color newColor)</code>	<p>Sets the pen color.</p> <p>You can specify the required color by defining RGB values or using a color value returned by any of the Color Queries [72]; for example:</p> <pre>GetUserFillColor()</pre> <p>Note:</p> <p>This method is only for line-drawing commands. It does not affect any text commands.</p>
<code>setpenwidth(</code> <code>int penwidth)</code>	<p>Sets the width of the pen. Pen width should be between 1 and 5.</p> <p>Note:</p> <p>This method is only for line-drawing commands. It does not affect any text commands.</p>
<code>showlabel(</code> <code>string labelname)</code>	Reveals the hidden label specified by labelname .
<code>startcloudpath(</code> <code>puffWidth,</code> <code>puffHeight,</code> <code>noise)</code>	<p>Similar to StartPath, except that it draws the path with cloud-like curved segments (<i>puffs</i>).</p> <p>Parameters:</p>

Method Name	Description
	<ul style="list-style-type: none"> • <i>float</i> puffWidth (default = 30), the horizontal distance between puffs • <i>float</i> puffHeight (default = 15), the vertical distance between puffs • <i>float</i> noise (default = 1.0), the randomization of the puffs' positions.
startpath()	Starts the sequence of drawing commands that define a path.
strokepath()	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](#)^[65] for IF statement syntax.
- [Query Methods](#)^[72] for methods that can be used as the conditional expression for IF statements.
- [Example Scripts](#)^[77] 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
boolean HasTag(string tagname, [string tagvalue])	Returns true 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 true .
boolean HasProperty(string propertyname, [string propertyvalue])	Returns true 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 true . See Display Item Properties ^[72] 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#");`

You can also test against and display an element's *custom* properties in the same way as you do the named properties.

For example: `if(hasproperty("Name", "Value"))`

...

and: `print("#Name#");`

Properties Visible to Shape Scripts

Properties for Element Shape Scripts

- addin (value returned from an Add-In)
- alias
- author
- cardinality
- classifier
- classifier.alias
- classifier.metatype
- classifier.stereotype
- classifier.type
- complexity
- concurrency
- datecreated
- datemodified
- diagram.mdgtype
- diagram.name
- diagram.stereotype
- diagram.type
- haslinkeddokument
- isabstract
- isactive
- iscomposite
- isembedded
- isinparent
- isleaf
- islocked
- isroot
- isspec
- istagged
- keywords
- language
- metatype
- multiplicity
- name
- notes
- packagename
- parentedge ("Right", "Left", "Top", "Bottom")
- parent.metatype
- persistence

- phase
- propertytype
- propertytype.alias
- propertytype.metatype
- propertytype.name
- propertytype.stereotype
- scope
- status
- stereotype
- type
- version
- visibility.

Properties for Connector Shape Scripts

- addin (value returned from an Add-In)
- 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* layout 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.

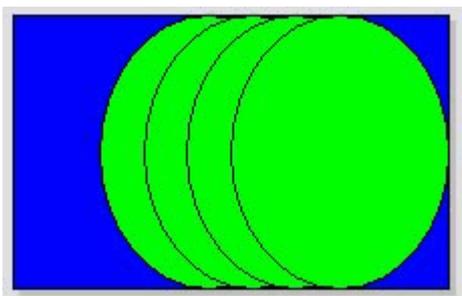
A subshape that is assigned to the *E* or *W* region must have its *preferredwidth* attribute specified in its declaration. Similarly, subshapes added to *N* or *S* must have their *preferredheight* attribute set. In this case, the values for these attributes are treated as static lengths and do not 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
shape main	The main shape is the whole shape.
shape label	The shape label gives the shape a detached label.
decoration <identifier>	Decoration 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
shape main	The main shape is the whole shape.
shape source	The source shape is an extra shape at the source end of the connector.
shape target	The target shape is an extra shape at the target end of the connector.
shape <labelID>	The <labelID> gives the connector a detached label, where <labelID> is one of the following: <ul style="list-style-type: none"> • LeftTopLabel • MiddleTopLabel • RightTopLabel • LeftBottomLabel • MiddleBottomLabel • RightBottomLabel

3.3.10 Miscellaneous

Return Command

Execution of the script can be terminated by using the return command. Please see [Example Scripts](#)  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

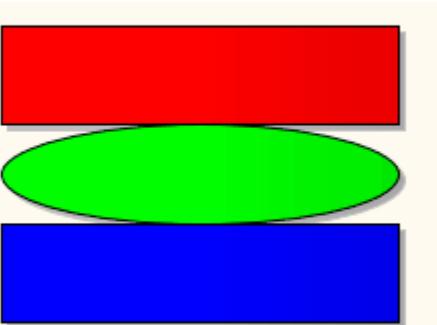
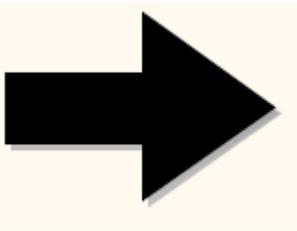
Not possible.

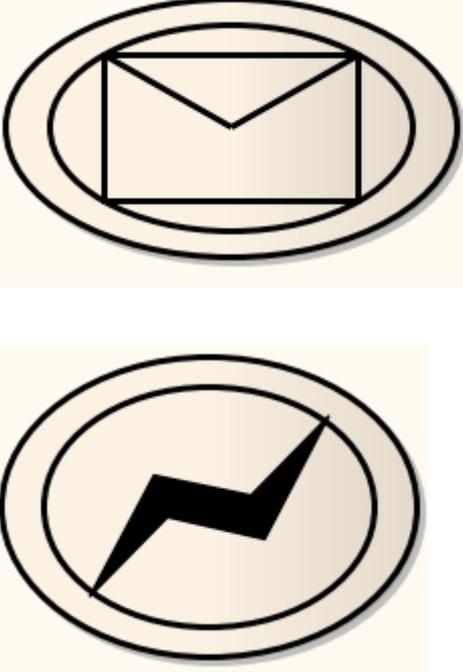
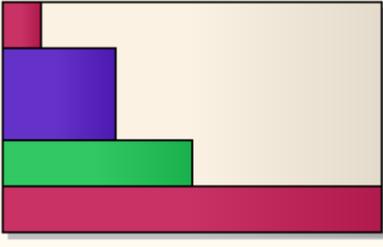
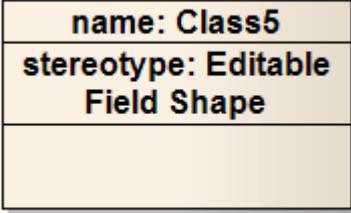
Can I apply a Shapascript without using Stereotypes?

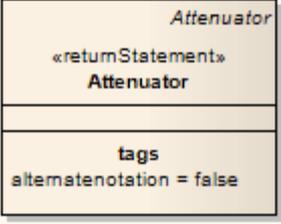
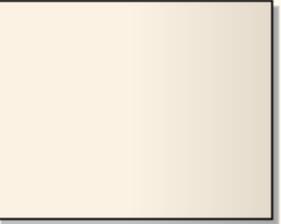
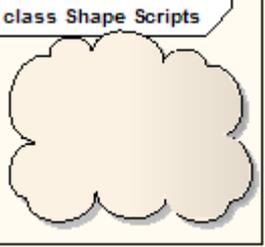
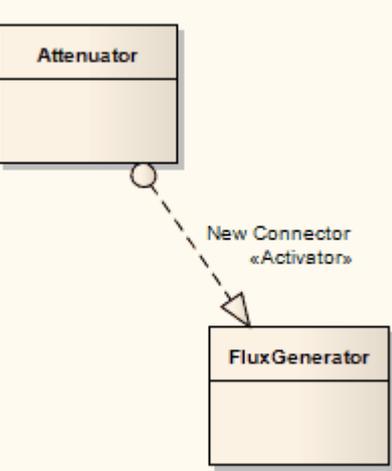
No.

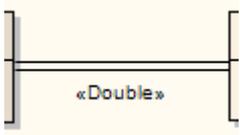
3.4 Example Scripts

Below is a selection of example Shape Scripts.

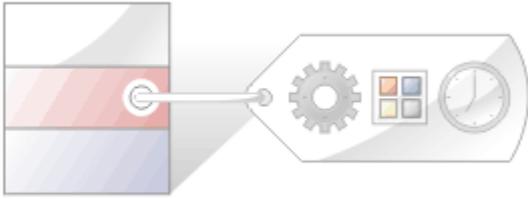
Code	Result
<pre>//BASIC SHAPES shape main { setfillcolor(255,0,0); // (R,G,B) rectangle(0,0,90,30); // (x1,y1,x2,y2) setfillcolor(0,255,0); // (R,G,B) ellipse(0,30,90,60); // (x1,y1,x2,y2) setfillcolor(0,0,255); // (R,G,B) rectangle(0,60,90,90); // (x1,y1,x2,y2) }</pre>	
<pre>//SINGLE CONDITIONAL SHAPE 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](#)^[81] with a specific format, with or without tag [filters](#)^[82], or
- Tagged Values that return values from the various [reference data](#)^[84] tables.

You can also use a masking parameter to create your own customized [masked Tagged Value Type](#)^[86].

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](#)^[83].

Note:

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

Tagged Value Type	Format	Description
Boolean	Type=Boolean; Default=Val;	Enables input of True or False , either of which can be the default value.
Classifier	Type=Classifier; Values=Type1,Type2; Stereotypes=Stereotype 1;	Deprecated. Use RefGUID and RefGUIDList. Returns the <i>name</i> of a user-selected element from the model, where Type1 and Type2 specify one or more allowed element types and Stereotype1 represents an allowed stereotype.
Color	Type=Color; Default=Val;	Enables input of a color value from a color chooser menu, where the value is the decimal integer translation of the color's Hex RGB value. For example, the RGB for Red is FF, and the decimal value is 255.
Const	Type=Const; Default=Val;	Enables creation of a read-only constant value.
Custom	Type= Custom;	Enables you to create your own template for predefined types; more information is provided in the Create Custom Tagged Value Type ^[86] topic.
DateTime	Type=DateTime;	Enables input of the date and time for the Tagged Value from a calendar menu.
Directory	Type=Directory; Default=Val;	Enables entry of a directory path from a browser. You can set a default directory path as a string value.

Tagged Value Type	Format	Description
Enum	Type=Enum; Values=Val1,Val2,Val3; Default=Val2;	Enables definition of a comma-separated list, where Val1 , Val2 and Val3 represent values in the list and Default represents the default value of the list.
File	Type=File; Default=Val;	Enables input of a filename from a file browser dialog. The named file can be launched in its default application. You can set a default file as a string containing the file path and file name.
Float, Decimal, Double	Type=Float; Type=Decimal; Type=Double; Default=Val;	Enable entry of a Float, Decimal or Double value. These types all map to the same type of data. You can set a default for any or all of these.
Integer	Type=Integer; Default=Val;	Enables entry of an Integer value, and a default.
Memo	Type=Memo;	Enables input of large and complex Tagged Values.
RefGUID	Type=RefGUID; Values=Type1,Type2; Stereotypes=Stereotype 1;	Enables the Tagged Value to reference an element from the model by specifying the element's <i>GUID</i> , where Type1 and Type2 specify one or more allowed diagram objects (such as Class , Component , Attribute or Operation) and Stereotype1 represents an allowed stereotype. Set the classifier (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i>), attribute or operation (see the <i>UML Dictionary</i>) for a Tagged Value of this type by clicking on the [...] button against the Tagged Value in the Tagged Value window.
RefGUIDList	Type=RefGUIDList; Values=Type1,Type2; Stereotypes=Stereotype 1;	Enables the Tagged Value to reference a list of elements from the model by specifying each element's <i>GUID</i> , where Type1 and Type2 specify one or more allowed diagram objects (such as Class or Component) and Stereotype1 represents an allowed stereotype. Set the classifier (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i>), attribute or operation (see the <i>UML Dictionary</i>) for a Tagged Value of this type by clicking on the [...] button against the Tagged Value in the Tagged Value window.
Spin	Type=Spin; LowerBound=x; UpperBound=x; Default=Val;	Enables creation of a spin control with the value of LowerBound being the lowest value and UpperBound being the highest value. You can also set a default within that range.
String	Type=String; Default=Val;	Enables entry of a string value, up to 255 characters in length, and a default text string. For longer texts, use <i>Type=Memo</i>  .

Tag Filters

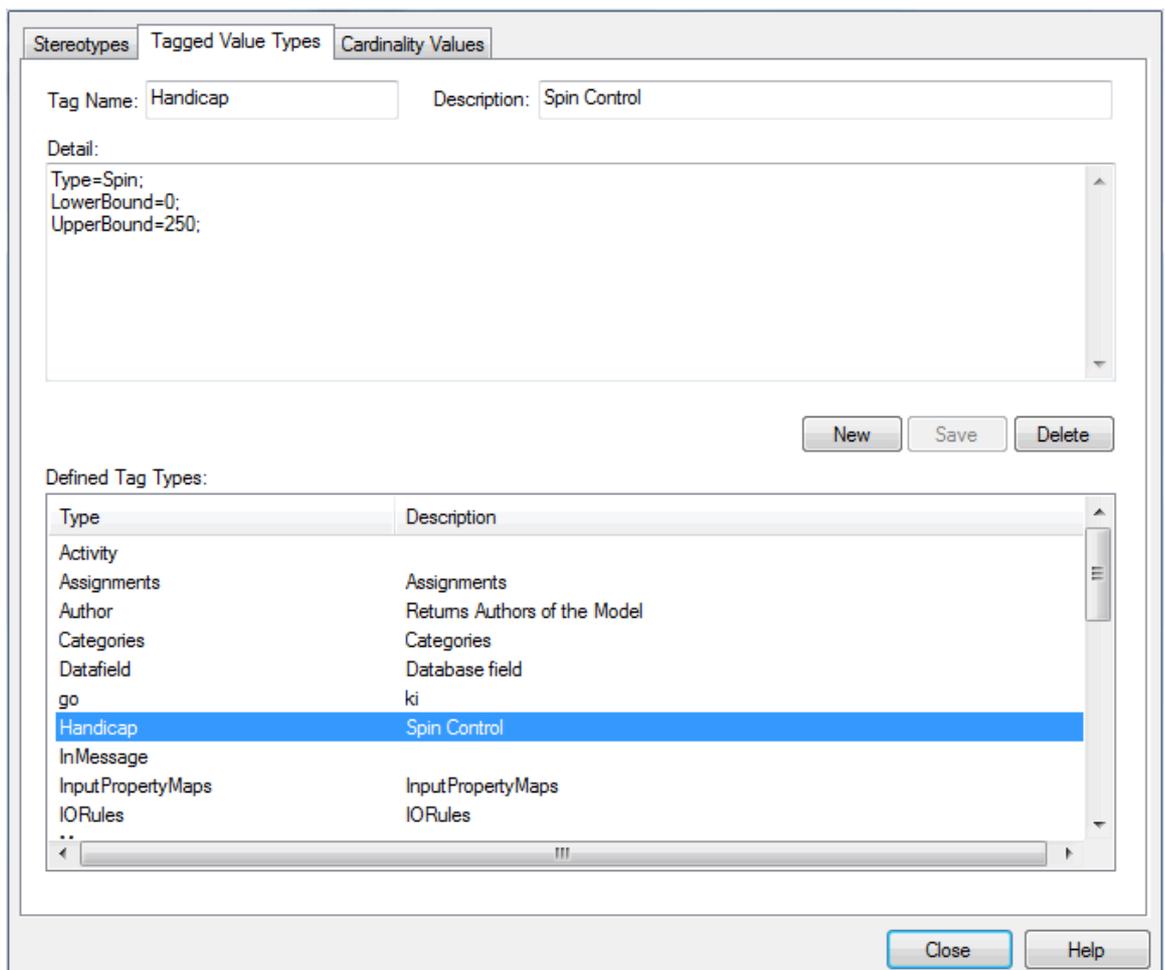
The following table details filters that can be used to restrict where a Tagged Value can be applied.

Filter	Format	Description
AppliesTo	AppliesTo=Type1, Type2;	Restricts the element types this filter can be applied to, where Type1 and Type2 are the valid types. Possible values are: <ul style="list-style-type: none"> • all element types • all connector types • Attribute • Operation and • OperationParameter.
BaseStereotype	BaseStereotype=S1,S2;	Restricts the stereotypes that this tag belongs to, where S1 and S2 are the allowed stereotypes.

4.2 Create Structured Tagged Values

To create your own Tagged Value based on a predefined [structured Tagged Value Type](#)^[81], follow the steps below:

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



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](#)⁸⁵.

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

Tagged Value Type	Format	Drop-Down List Returned
	List=Resources;	Architect model.
RiskTypes	Type=Enum; List=RiskTypes;	Risk types that have been defined for the Enterprise Architect model.
RTFTemplates	Type=Enum; List=RTFTemplates;	RTF Templates that have been defined for the Enterprise Architect model.
ScenarioTypes	Type=Enum; List=ScenarioTypes;	Scenario types that have been defined for the Enterprise Architect model.
TestTypes	Type=Enum; List=TestTypes;	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](#)^[84], follow the steps below:

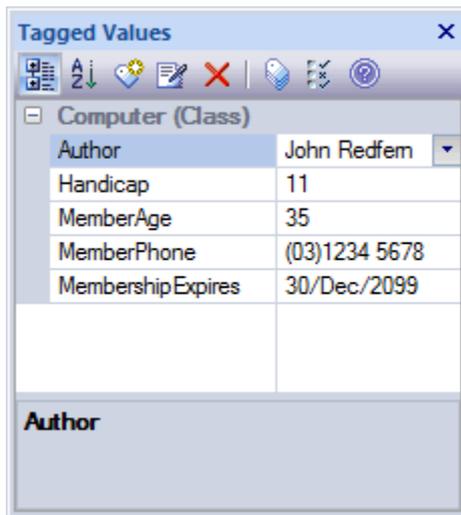
1. Select the **Settings | UML** menu option. The **UML Types** dialog displays; select the **Tagged Value Types** tab.

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

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*).

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

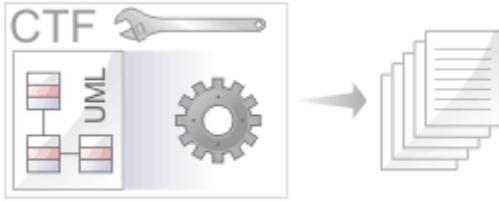
With the **Template** configuration option, the syntax defines the template of the masked option by occupying the blank spaces that are present in the **Mask** option. The template is used to ensure that this information is

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.

The screenshot shows a software interface for configuring Tagged Value Types. It has three tabs: 'Stereotypes', 'Tagged Value Types' (which is active), and 'Cardinality Values'. In the 'Tagged Value Types' tab, there are two input fields: 'Tag Name' containing 'MemberZip' and 'Description' containing 'Zip Code'. Below these is a 'Detail' section with a scrollable text area containing the following text: 'Type=Custom:', 'Mask= cc dddd.dddd;', and 'Template=State: __Zip: _____-____:'. To the right of the 'Detail' section are three buttons: 'New', 'Save', and 'Delete'. At the bottom, there is a table titled 'Defined Tag Types' with two columns: 'Type' and 'Description'. The table contains the following rows: 'IORules' with description 'IORules', 'MemberZip' with description 'Zip Code' (highlighted in blue), 'Message', and 'OutMessage'.

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](#).

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](#) 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](#)). The template syntax centers on three basic constructs:

- [Literal Text](#)
- [Macros](#)
- [Variables](#)

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
%d%	Produce a literal \$ character.
%pc%	Produce a literal % character.
%qt%	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](#) ^[89]
- [Field substitution macros](#) ^[90]
- [Tagged Value substitution macros](#) ^[102]
- [Control macros](#) ^[106]
- [Function macros](#) ^[103]
- [EASL code generation macros](#) ^[109]

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	Attributes dialog: Alias .
attAllowDuplicates	Attributes Detail dialog: Allow Duplicates checkbox.
attClassifierGUID	The unique GUID for the classifier of the current attribute.
attCollection	Attributes Detail dialog: Attribute is a Collection checkbox.
attConst	Attributes dialog: Const checkbox.
attContainerType	Attributes Detail dialog: Container Type .
attContainment	Attributes dialog: Containment .
attDerived	Attributes dialog: Derived checkbox.
attGUID	The unique GUID for the current attribute.
attInitial	Attributes dialog: Initial .
attIsEnumLiteral	Attributes dialog: Is Literal checkbox.
attLength	Column dialog: Length .
attLowerBound	Attributes Detail dialog: Lower Bound .
attName	Attributes dialog: Name .
attNotes	Attributes dialog: Notes .
attOrderedMultiplicity	Attributes Detail dialog: Ordered Multiplicity checkbox.
attProperty	Attributes dialog: Property 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	Attributes dialog: Scope .
attStatic	Attributes dialog: Static checkbox.

Macro Name	Description
attStereotype	Attributes dialog: Stereotype .
attType	Attributes dialog: Type .
attUpperBound	Attributes Detail dialog: Upper Bound .
attVolatile	Attributes Detail dialog: Transient checkbox.
classAbstract	Class dialog: Abstract checkbox.
classAlias	Class dialog: Alias .
classArguments	Class Detail dialog: C++ Templates: Arguments .
classAuthor	Class dialog: Author .
classBaseName	Type Hierarchy dialog: Class Name (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	Class dialog: Complexity .
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 T if one is a default constructor. Typically used with the genOptGenConstructor ⁹⁶ macro.
classHasCopyConstructor	Looks at the list of methods in the current object and, depending on the conventions of the current language, returns T if one is a copy constructor. Typically used with the genOptGenCopyConstructor ⁹⁶ macro.
classHasDestructor	Looks at the list of methods in the current object and, depending on the conventions of the current language, returns T if one is a destructor. Typically used with the genOptGenDestructor ⁹⁶ macro.
classHasParent	True , if the Class in scope has one or more base Classes.
classImports	Code Gen dialog: Imports .
classIsActive	Class Advanced dialog: Is Active checkbox.
classIsInstantiated	True , if the Class is an instantiated template Class.
classIsLeaf	Class Advanced dialog: Is Leaf checkbox.
classIsRoot	Class Advanced dialog: Is Root checkbox.
classIsSpecification	Class Advanced dialog: Is Specification checkbox.
classKeywords	Class dialog: Keywords .
classLanguage	Class dialog: Language .
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	Class Advanced dialog: Multiplicity .
className	Class dialog: Name .
classNotes	Class dialog: Note .
classParamDefault	Class Detail dialog.
classParamName	Class Detail dialog.
classParamType	Class Detail dialog.
classPersistence	Class dialog: Persistence .
classPhase	Class dialog: Phase .
classQualName	The Class name prefixed by its outer Classes. Class names are separated by double colons (::).
classScope	Class dialog: Scope .
classStereotype	Class dialog: Stereotype .
classStatus	Class dialog: Status .
classVersion	Class dialog: Version .
connectorAlias	Connector Properties dialog: Alias .
connectorDestAccess	Connector Properties dialog, Target Role tab: Access .
connectorDestAggregation	Connector Properties dialog, Target Role tab: Aggregation .
connectorDestAlias	Connector Properties dialog, Target Role tab: Alias .
connectorDestAllowDuplicates	Connector Properties dialog, Target Role tab: Allow Duplicates checkbox.
connectorDestChangeable	Connector Properties dialog, Target Role tab: Changeable .
connectorDestConstraint	Connector Properties dialog, Target Role tab: Constraint(s) .
connectorDestContainment	Connector Properties dialog, Target Role tab: Containment .
connectorDestDerived	Connector Properties dialog, Target Role tab: Derived checkbox.
connectorDestDerivedUnion	Connector Properties dialog, Target Role tab: DerivedUnion 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 Properties dialog, Target Role tab: Member Type .
connectorDestMultiplicity	Connector Properties dialog, Target Role tab: Multiplicity .

Macro Name	Description
connectorDestNavigability	Connector Properties dialog, Target Role tab: Navigability .
connectorDestNotes	Connector Properties dialog, Target Role tab: Role Notes .
connectorDestOrdered	Connector Properties dialog, Target Role tab: Ordered checkbox.
connectorDestOwned	Connector Properties dialog, Target Role tab: Owned checkbox.
connectorDestQualifier	Connector Properties dialog, Target Role tab: Qualifier(s) .
connectorDestRole	Connector Properties dialog, Target Role tab: Role .
connectorDestScope	Connector Properties dialog, Target Role tab: Target Scope .
connectorDestStereotype	Connector Properties dialog, Target Role tab: Stereotype .
connectorDirection	Connector Properties : Direction .
connectorEffect	Transition Constraints dialog: Effect .
connectorGuard	Object Flow and Transition Constraints dialog: Guard .
connectorGUID	The unique GUID for the current connector.
connectorName	Connector Properties : Name .
connectorNotes	Connector Properties : Notes .
connectorSourceAccess	Connector Properties dialog, Source Role tab: Access .
connectorSourceAggregation	Connector Properties dialog, Source Role tab: Aggregation .
connectorSourceAlias	Connector Properties dialog, Source Role tab: Alias .
connectorSourceAllowDuplicates	Connector Properties dialog, Source Role tab: Allow Duplicates checkbox.
connectorSourceChangeable	Connector Properties dialog, Source Role tab: Changeable .
connectorSourceConstraint	Connector Properties dialog, Source Role tab: Constraint(s) .
connectorSourceContainment	Connector Properties dialog, Source Role tab: Containment .
connectorSourceDerived	Connector Properties dialog, Source Role tab: Derived checkbox.
connectorSourceDerivedUnion	Connector Properties dialog, Source Role tab: DerivedUnion 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 Properties dialog, Source Role tab: Member Type .
connectorSourceMultiplicity	Connector Properties dialog, Source Role tab: Multiplicity .
connectorSourceNavigability	Connector Properties dialog, Source Role tab: Navigability .

Macro Name	Description
connectorSourceNotes	Connector Properties dialog, Source Role tab: Role Notes .
connectorSourceOrdered	Connector Properties dialog, Source Role tab: Ordered checkbox.
connectorSourceOwned	Connector Properties dialog, Source Role tab: Owned checkbox.
connectorSourceQualifier	Connector Properties dialog, Source Role tab: Qualifier(s) .
connectorSourceRole	Connector Properties dialog, Source Role tab: Role .
connectorSourceScope	Connector Properties dialog, Source Role tab: Target Scope .
connectorSourceStereotype	Connector Properties dialog, Source Role tab: Stereotype .
connectorStereotype	Connector Properties dialog: Stereotype .
connectorTrigger	Transition Constraints dialog: Trigger .
connectorType	The connector type; for example, Association or Generalization.
connectorWeight	Object Flow Constraints dialog: Weight .
constraintName	Class dialog, Constraints tab: Name .
constraintNotes	Class dialog, Constraints tab: Notes .
constraintStatus	Class dialog, Constraints tab: Status .
constraintType	Class dialog, Constraints tab: Type .
constraintWeight	Class dialog, Constraints 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 Help About EA).
effortName	Project Management window: Effort .
effortNotes	Project Management window: Notes (unlabelled).
effortTime	Project Management window: Time .
effortType	Project Management window: Type .
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	Code Gen dialog: Headers .
fileImports	Code Gen dialog: Imports . 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: Default Version.
genOptCDefaultAttributeType	C Specifications dialog: Default Attribute Type.
genOptCGenMethodNotesInBody	C Specifications dialog: Method Notes In Implementation.
genOptCGenMethodNotesInHeader	C Specifications dialog: Method Notes In Header.
genOptCSynchNotes	C Specifications dialog: Synchronize Notes in Generation.
genOptCSynchCFile	C Specifications dialog: Synchronise Implementation file in Generation.
genOptCDefaultSourceDirectory	C Specifications dialog: Default Source Directory.
genOptCNamespaceDelimiter	C Specifications dialog: Namespace Delimiter.
genOptCOperationRefParam	C Specifications dialog: Reference as Operation Parameter.
genOptCOperationRefParamStyle	C Specifications dialog: Reference Parameter Style.
genOptCOperationRefParamName	C Specifications dialog: Reference Parameter Name.
genOptCConstructorName	C Specifications dialog: Default Constructor Name.
genOptCDestructorName	C Specifications dialog: Default Destructor Name.
genOptCPPCommentStyle	C++ Specifications dialog: Comment Style.
genOptCPPDefaultAttributeType	C++ Specifications dialog: Default Attribute Type.
genOptCPPDefaultReferenceType	C++ Specifications dialog: Default Reference Type.
genOptCPPDefaultSourceDirectory	C++ Specifications dialog: Default Source Directory.
genOptCPPGenMethodNotesInHeader	C++ Specifications dialog: Method Notes In Header checkbox.
genOptCPPGenMethodNotesInBody	C++ Specifications dialog: Method Notes In Body checkbox.
genOptCPPGetPrefix	C++ Specifications dialog: Get Prefix.
genOptCPPHeaderExtension	C++ Specifications dialog: Header Extension.
genOptCPPSetPrefix	C++ Specifications dialog: Set Prefix.
genOptCPPSourceExtension	C++ Specifications dialog: Source Extension.
genOptCPPSynchCPPFile	C++ Specifications dialog: Synchronize Notes.
genOptCPPSynchNotes	C++ Specifications dialog: Synchronize CPP File.
genOptCSDefaultAttributeType	C# Specifications dialog: Default Attribute Type.
genOptCSSourceExtension	C# Specifications dialog: Default file extension.
genOptCSGenDispose	C# Specifications dialog: Generate Dispose.
genOptCSGenFinalizer	C# Specifications dialog: Generate Finalizer.

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

Macro Name	Description
genOptPHPSourceExtension	PHP Specifications dialog: Default file extension.
genOptPHPVersion	PHP Specifications dialog: PHP Version.
genOptPropertyPrefix	Source Code Engineering dialog: Remove prefixes when generating Get/Set properties.
genOptVBMultiUse	VB Specifications dialog: Multiuse checkbox.
genOptVBPersistable	VB Specifications dialog: Persistable checkbox.
genOptVBDataBindingBehavior	VB Specifications dialog: Data binding behavior checkbox.
genOptVBDataSourceBehavior	VB Specifications dialog: Data source behavior checkbox.
genOptVBGlobal	VB Specifications dialog: Global namespace checkbox.
genOptVBCreatable	VB Specifications dialog: Creatable checkbox.
genOptVBExposed	VB Specifications dialog: Exposed checkbox.
genOptVBMTS	VB Specifications dialog: MTS Transaction Mode.
genOptVBNetGenNamespace	VB.Net Specifications dialog: Generate Namespace.
genOptVBVersion	VB Specifications dialog: Default Version.
genOptWrapComment	Source Code Engineering dialog: Wrap length for comment lines.
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	T if the Class has an Aggregation connector to a Class in this file, F otherwise.
importFromAssociation	T if the Class has an Association connector to a Class in this file, F otherwise.
importFromAtt	T if an attribute of a Class in the current file is of the type of this Class, F otherwise.
importFromDependency	T if the Class has a Dependency connector to a Class in this file, F otherwise.
importFromGeneralization	T if the Class has a Generalization connector to a Class in this file, F otherwise.
importFromMeth	T if a method return type of a Class in the current file is the type of this Class, F otherwise.
importFromParam	T if an method parameter of a Class in the current file is of the type of this Class, F otherwise.
importFromRealization	T if the Class has a Realization connector to a Class in this file, F otherwise.
importInFile	T if the Class is in the current file, F 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: Access .
linkAttCollectionClass	The collection appropriate for the linked attribute in scope.
linkAttContainment	Association Properties Target Role dialog: Containment .
linkAttName	Association Properties dialog: Target .
linkAttNotes	Association Properties Target Role dialog: Role Notes .
linkAttQualName	The Association target qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited).
linkAttRole	Association Properties Target Role dialog: Role .
linkAttStereotype	Association Properties Target Role dialog: Stereotype .
linkAttTargetScope	Association Properties Target Role dialog: Target Scope .
linkCard	Link Properties Target Role dialog: Multiplicity .
linkedFileLastWrite	Class Properties dialog: Last Write .
linkedFileNotes	Class Properties dialog: Notes .
linkedFilePath	Class Properties dialog: File Path .
linkedFileSize	Class Properties dialog: Size .
linkedFileType	Class Properties dialog: Type .
linkGUID	The unique GUID for the current connector.
linkParentName	Generalization Properties dialog: Target .
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: Virtual Inheritance .
metricName	Project Management dialog, Metrics tab: Metric field.
metricNotes	Project Management dialog, Metrics tab: (Notes) field.
metricType	Project Management dialog, Metrics tab: Type field.
metricWeight	Project Management dialog, Metrics tab: Weight field.
opAbstract	Operation dialog: Virtual checkbox.
opAlias	Operation dialog: Alias .
opBehavior	Operation Behavior dialog: Behavior .
opCode	Operation Behavior dialog: Initial Code .
opConcurrency	Operation dialog: Concurrency .

Macro Name	Description
opConst	Operation dialog: Const checkbox.
opGUID	The unique GUID for the current operation.
opImplMacros	A space-separated list of macros defined in the implementation of this operation.
opIsQuery	Operation dialog: IsQuery checkbox.
opMacros	A space-separated list of macros defined in the declaration for this operation.
opName	Operation dialog: Name .
opNotes	Operation dialog: Notes .
opPure	Operation dialog: Pure checkbox.
opReturnArray	Operation dialog: Return Array 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	Operation dialog: Return Type .
opScope	Operation dialog: Scope .
opStatic	Operation dialog: Static checkbox.
opStereotype	Operation dialog: Stereotype .
opSynchronized	Operation dialog: Synchronized checkbox.
packageAbstract	Package dialog: Abstract .
packageAlias	Package dialog: Alias .
packageAuthor	Package dialog: Author .
packageComplexity	Package dialog: Complexity .
packageGUID	The unique GUID for the current package.
packageKeywords	Package dialog: Keywords .
packageLanguage	Package dialog: Language .
packageName	Package dialog: Name .
packagePath	The string representing the hierarchy of packages, for the Class in scope. Each package name is separated by a dot (.).
packagePhase	Package dialog: Phase .
packageScope	Package dialog: Scope .
packageStatus	Package dialog: Status .
packageStereotype	Package dialog: Stereotype .

Macro Name	Description
packageVersion	Package dialog: Version .
paramClassifierGUID	The unique GUID for the classifier of the current parameter.
paramDefault	Operation Parameters dialog: Default .
paramFixed	Operation Parameters dialog: Fixed checkbox.
paramGUID	The unique GUID for the current parameter.
paramsEnum	True , if the parameter uses the <i>enum</i> keyword (C++).
paramKind	Operation Parameters dialog: Kind .
paramName	Operation Parameters dialog: Name .
paramNotes	Operation Parameters dialog: Notes .
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: Type .
problemCompletedBy	Maintenance dialog, Element Issues tab: Completed by .
problemCompletedDate	Maintenance dialog, Element Issues tab: Completed .
problemHistory	Maintenance dialog, Element Issues tab: History .
problemName	Maintenance dialog, Element Issues tab: Name .
problemNotes	Maintenance dialog, Element Issues tab: Description .
problemPriority	Maintenance dialog, Element Issues tab: Priority .
problemRaisedBy	Maintenance dialog, Element Issues tab: Raised by .
problemRaisedDate	Maintenance dialog, Element Issues tab: Raised .
problemStatus	Maintenance dialog, Element Issues tab: Status .
problemVersion	Maintenance dialog, Element Issues tab: Version .
requirementDifficulty	Properties dialog: Require tab: Difficulty .
requirementLastUpdated	Properties dialog: Require tab: Last Update .
requirementName	Properties dialog: Require tab: Short Description .
requirementNotes	Properties dialog: Require tab: Notes .
requirementPriority	Properties dialog: Require tab: Priority .
requirementStatus	Properties dialog: Require tab: Status .
requirementType	Properties dialog: Require tab: Type .
resourceAllocatedTime	Project Management window, Resource Allocation tab: Allocated Time .
resourceEndDate	Project Management window, Resource Allocation tab: End Date .
resourceExpectedTime	Project Management window, Resource Allocation tab: Expected Time .

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

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`
- `connectorDestElemTag`

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

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

`<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](#).^[106]

PROCESS_END_OBJECT(<template_name>)

Enables the Classes that are one Class further away from the base Class, to be transformed into objects (such as attributes, operations, packages, parameters and columns) of the base Class. <template_name> refers to the working template that temporarily stores the data.

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](#)^[103] 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](#))^[119].
- 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)%
```

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

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

```
%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
synchNewClassNotesSpace	Space to append to a new Class note. Default value: \n.
synchNewAttributeNotesSpace	Space to append to a new attribute note. Default value: \n.
synchNewOperationNotesSpace	Space to append to a new operation note. Default value: \n.
synchNewOperationBodySpace	Space to append to a new operation body. Default value: \n.
synchNamespaceBodyIndent	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](#)^[117] and [EASL Properties](#)^[113] 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](#)^[110] or [custom template](#)^[119]
- <<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

```
$sStates = %EASLList="State" @separator="\n" @indent="\t"
           @owner=$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](#)^[109] 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](#)^[109] 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](#)^[117] and [referenced](#)^[118].

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](#) ^[119]
- [Override Default Templates](#) ^[120]
- [Add New Stereotyped Templates](#) ^[121]
- [Create Templates For Custom Languages](#) ^[122]

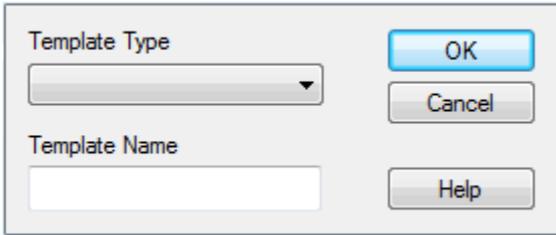
The Code Template Editor provides the facilities of the *Common Code Editor*, including intellisense for the code generation template [macros](#) ^[89]. For more information on intellisense and the Common Code Editor, see the *Code Editors* topic in *Using Enterprise Architect - UML Modeling Tool*.

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.

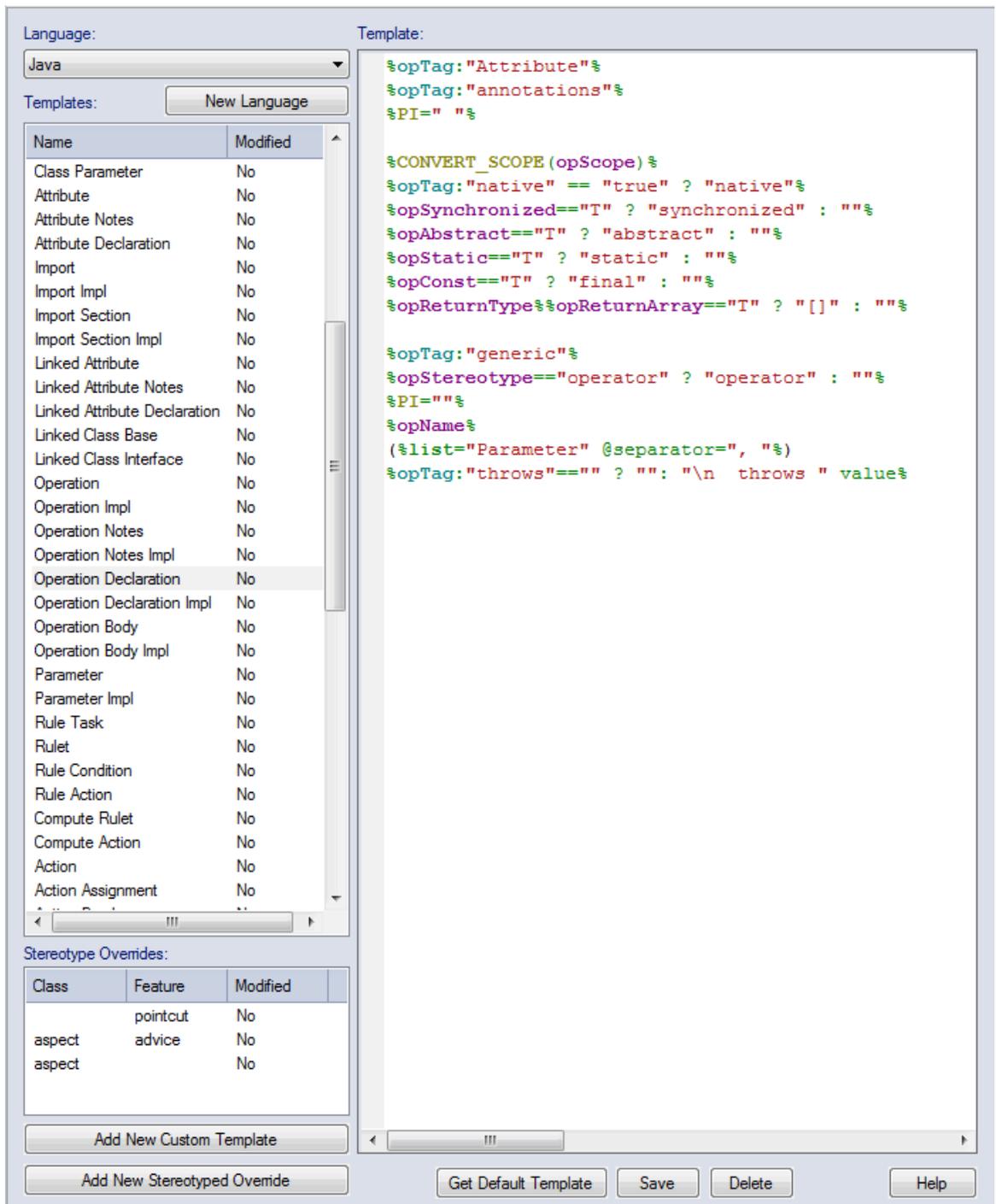


4. In the **Template Type** field, click on the drop-down arrow and select the appropriate element. The elements currently supported are:
 - Attribute
 - Class
 - Class Base
 - Class Interface
 - Class Parameter
 - Connector
 - Import
 - Linked Attribute
 - Linked Class Base
 - Linked Class Interface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

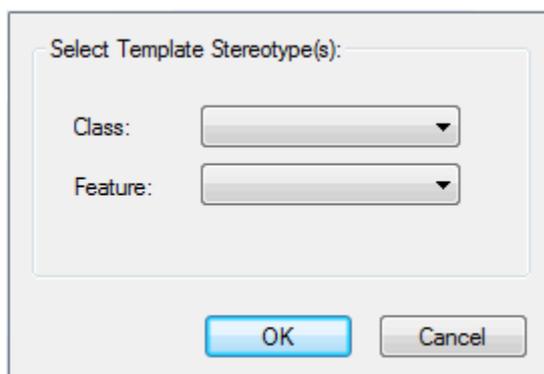
5.2.3 Add New Stereotyped Templates

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.

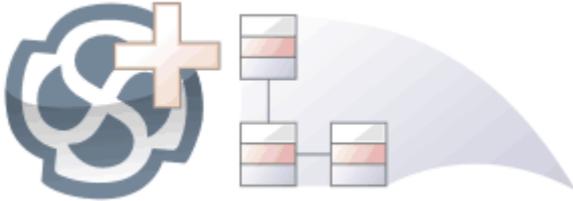
Define a Template for a Custom Language

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](#) ^[187] 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](#) ^[37] and the [Automation Interface](#) ^[187]. Once an Add-In is registered (see *Getting Started With Enterprise Architect*), it can be managed using the [Add-In Manager](#) ^[128].

Create and Use Add-Ins

This topic covers the following information on Add-Ins:

- [Add-In Tasks](#) ^[124]
- [Add-In Events](#) ^[129]
- [Broadcast Events](#) ^[135]
- [Custom Views](#) ^[169]
- [MDG Add-Ins](#) ^[170]

6.1 Add-In Tasks

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

1. [Create an Add-In](#)^[124]
 - [Define Menu Items](#)^[124]
 - [Respond to Menu Events](#)^[132]
 - [Handle Add-In Events](#)^[129]
2. [Deploy your Add-In](#)^[125]
 - [Potential Pitfalls](#)^[126]
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](#)^[128]

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](#)^[124]. 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](#)^[181] topic.
3. Create a COM Class and implement each of the [general Add-In Events](#)^[129] 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](#)^[125] 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:

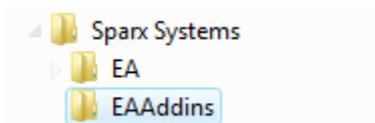
1. Add the Add-In DLL file to an appropriate directory on the user's computer; that is, C:\Program Files\[new dir].
2. Register the DLL as appropriate to your platform:
 - If compiled as a native Win32 DDL, such as VB6 or C++, register the DDL using the **regsvr32** command from the command prompt; for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

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

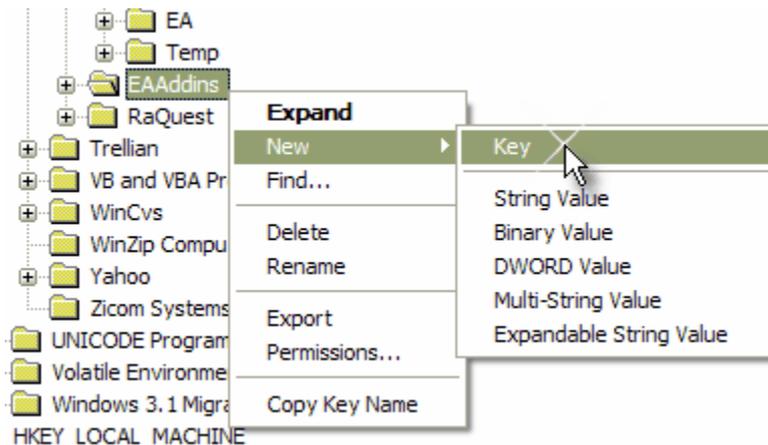
- If compiled as a .NET DLL, such as C# or VB.NET, register the DLL using the **RegAsm** command from the command prompt; for example:

```
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

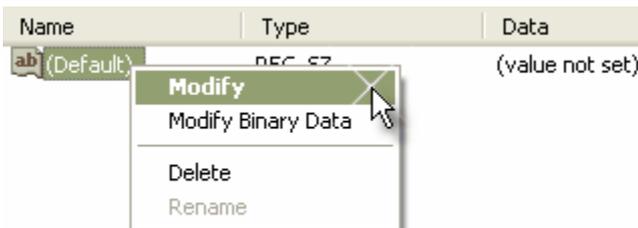


5. Add a new key under this key with the project name.

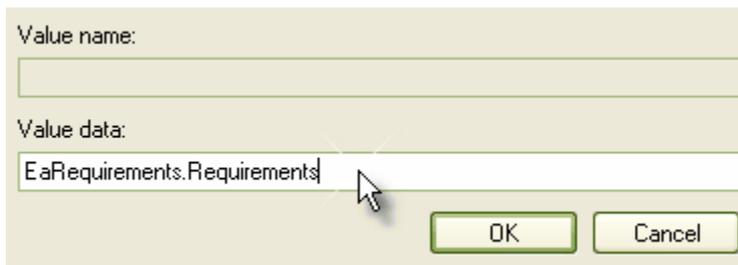
**Note:**

[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](#)^[184] 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](#)^[127] 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](#)^[129] 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 [Manage Add-Ins](#) 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
Rep	The currently open repository.
SearchText	An optional field that you can fill in through the Model Search .
XMLResults	At completion of the method, this should contain the results for the search. The results should be an XML String that conforms to the Search Data Format ^[129] .

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](#)^[197] 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](#)^[130]
- [EA_Disconnect](#)^[130]
- [EA_GetMenuItems](#)^[131]

- [EA_MenuClick](#)^[132]
- [EA_GetMenuState](#)^[131]
- [EA_ShowHelp](#)^[134]
- [EA_OnOutputItemClicked](#)^[133]
- [EA_OnOutputItemDoubleClicked](#)^[133]

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](#)^[170].

Also look at [EA_Disconnect](#)^[130].

Syntax

Function *EA_Connect(Repository As EA.Repository) As String*

The *EA_Connect* function syntax has the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository ^[197]	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 MDG Events ^[170] 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](#)^[130].

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](#)^[124] topic.

Also look at:

- [EA_MenuClick](#)^[132]
- [EA_GetMenuState](#)^[131].

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
MenuLocation	<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> .
MenuName	<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.
Repository	EA.Repository ^[197]	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](#)^[124] topic.

Also look at [EA_GetMenuItems](#)^[131].

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
IsChecked	<i>Boolean</i>		Boolean. Set to True to check this particular menu option.
IsEnabled	<i>Boolean</i>		Boolean. Set to False to disable this particular menu option.
ItemName	<i>String</i>		The name of the option actually clicked, for example, <i>Create a New Invoice</i> .
MenuLocation	<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> .
MenuName	<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.
Repository	EA.Repository ^[197]	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](#)^[196] methods.

Also look at [EA_GetMenuItems](#)^[137].

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
ItemName	<i>String</i>		The name of the option actually clicked, for example, <i>Create a</i>

Parameter	Type	Direction	Description
			<i>New Invoice.</i>
MenuName	<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.
Repository	EA.Repository <small>[197]</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](#)^[133].

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
ID	<i>Long</i>	IN	The ID value specified in the original call to <i>Repository.WriteOutput()</i> .
LineText	<i>String</i>	IN	The text that had been supplied as the String parameter in the original call to <i>Repository.WriteOutput()</i> .
Repository	EA.Repository <small>[197]</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.
TabName	<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](#)^[133].

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
ID	<i>Long</i>	IN	The ID value specified in the original call to <i>Repository.WriteOutput()</i> .
LineText	<i>String</i>	IN	The text that had been supplied as the String parameter in the original call to <i>Repository.WriteOutput()</i> .
Repository	EA.Repository ^[197]	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.
TabName	<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](#)^[134].

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
ItemName	<i>String</i>		The name of the option actually clicked, for example, Create a New Invoice .
MenuLocation	<i>String</i>		String representing the part of the user interface that brought up the menu. Can be Treeview , MainMenu or Diagram .
MenuName	<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	EA.Repository ^[197]	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 to the Add-In.

- [File Open Event](#)^[135]
- [File Close Event](#)^[136]
- [File New Event](#)^[136]
- [Post Diagram Open Event](#)^[137]
- [Post Diagram Close Event](#)^[137]
- [Pre-Deletion Events](#)^[138]
- [Pre-New Events](#)^[141]
- [Post-New Events](#)^[146]
- [Technology Events](#)^[152]
- [Context Item Events](#)^[155]
- [Transformation Events](#)^[151]
- [Compartment Events](#)^[158]
- [Model Validation Broadcasts](#)^[160]
- [Retrieve Model Template Event](#)^[168]
- [Initialize Technology Event](#)^[152]
- [Exit Instance](#)^[146] (not currently used).

[MDG Events](#)^[170] 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](#)^[136] and [EA_FileNew](#)^[136].

Syntax

Sub EA_FileOpen(Repository As EA.Repository)

The *EA_FileOpen* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository <small>[197]</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.

6.5.2 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](#)^[135] and [EA_FileNew](#)^[136].

Syntax

Sub EA_FileClose(Repository As EA.Repository)

The *EA_FileClose* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository <small>[197]</small>	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](#)^[135] and [EA_FileOpen](#)^[136].

Syntax

Sub EA_FileNew(Repository As EA.Repository)

The *EA_FileNew* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository <small>[197]</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.

6.5.4 EA_OnPostCloseDiagram

Details

EA_OnPostCloseDiagram notifies Add-Ins that a diagram has been closed.

Also look at [EA_OnPostOpenDiagram](#)^[137].

Syntax

Function *EA_OnPostCloseDiagram(Repository As EA.Repository, DiagramID As Integer)*

The *EA_OnPostCloseDiagram* function syntax contains the following elements:

Parameter	Type	Direction	Description
DiagramID	<i>Integer</i>	IN	Contains the Diagram ID of the diagram that was closed.
Repository	EA.Repository ^[197]	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.

6.5.5 EA_OnPostOpenDiagram

Details

EA_OnPostOpenDiagram notifies Add-Ins that a diagram has been opened.

Also look at [EA_OnPostCloseDiagram](#)^[137].

Syntax

Function *EA_OnPostOpenDiagram(Repository As EA.Repository, DiagramID As Integer)*

The *EA_OnPostOpenDiagram* function syntax contains the following elements:

Parameter	Type	Direction	Description
DiagramID	<i>Integer</i>	IN	Contains the Diagram ID of the diagram that was opened.
Repository	EA.Repository ^[197]	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.

6.5.6 Pre-Deletion Events

Enterprise Architect Add-Ins can respond to requests to delete elements, attributes, methods, connectors, diagrams, packages and technologies using the following broadcast events:

- [EA_OnPreDeleteElement](#)^[138]
- [EA_OnPreDeleteAttribute](#)^[138]
- [EA_OnPreDeleteMethod](#)^[139]
- [EA_OnPreDeleteConnector](#)^[139]
- [EA_OnPreDeleteDiagram](#)^[140]
- [EA_OnPreDeletePackage](#)^[141]
- [EA_OnPreDeleteTechnology](#)^[153] (*Deprecated*).

6.5.6.1 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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the element to be deleted: <ul style="list-style-type: none"> • <i>ElementID</i>: A long value corresponding to <i>Element.ElementID</i>.
Repository	EA.Repository ^[197]	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.

6.5.6.2 EA_OnPreDeleteAttribute

Details

EA_OnPreDeleteAttribute notifies Add-Ins that an attribute is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the attribute.

This event occurs when a user attempts to permanently delete an attribute from the **Project Browser**. The notification is provided immediately before the attribute is deleted, so that the Add-In can disable deletion of the attribute.

Syntax

Function *EA_OnPreDeleteAttribute(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA_OnPreDeleteAttribute* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	EA.EventProperties <small>[214]</small>	IN	Contains the following <i>EventProperty Objects</i> for the attribute to be deleted: <ul style="list-style-type: none"> <i>AttributeID</i>: A long value corresponding to <i>Attribute.AttributeID</i>.
Repository	EA.Repository <small>[197]</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 attribute from the model. Return **False** to disable deletion of the attribute.

6.5.6.3 EA_OnPreDeleteMethod

Details

EA_OnPreDeleteMethod notifies Add-Ins that a method (operation) is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the method.

This event occurs when a user attempts to permanently delete a method from the **Project Browser**. The notification is provided immediately before the method is deleted, so that the Add-In can disable deletion of the method.

Syntax

Function *EA_OnPreDeleteMethod(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA_OnPreDeleteMethod* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	EA.EventProperties <small>[214]</small>	IN	Contains the following <i>EventProperty Objects</i> for the method to be deleted: <ul style="list-style-type: none"> <i>MethodID</i>: A long value corresponding to <i>Method.MethodID</i>.
Repository	EA.Repository <small>[197]</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 method from the model. Return **False** to disable deletion of the method.

6.5.6.4 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	EA.EventProperties <small>[214]</small>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <i>ConnectorID</i>: A long value corresponding to <i>Connector.ConnectorID</i>.
Repository	EA.Repository <small>[197]</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.

6.5.6.5 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	EA.EventProperties <small>[214]</small>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <i>DiagramID</i>: A long value corresponding to <i>Diagram.DiagramID</i>.
Repository	EA.Repository <small>[197]</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.

6.5.6.6 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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <i>PackageID</i>: A long value corresponding to <i>Package</i>. <i>PackageID</i>.
Repository	EA.Repository ^[197]	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.

6.5.7 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](#)^[141]
- [EA_OnPreNewConnector](#)^[142]
- [EA_OnPreNewDiagram](#)^[143]
- [EA_OnPreNewDiagramObject](#)^[143]
- [EA_OnPreNewAttribute](#)^[144]
- [EA_OnPreNewMethod](#)^[145]
- [EA_OnPreNewPackage](#)^[145].

6.5.7.1 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](#)^[147].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the element to be created: <ul style="list-style-type: none"> <i>Type</i>: A string value corresponding to <i>Element.Type</i> <i>Stereotype</i>: A string value corresponding to <i>Element.Stereotype</i> <i>ParentID</i>: A long value corresponding to <i>Element.ParentID</i> <i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the element is being added.
Repository	EA.Repository ^[197]	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.

6.5.7.2 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](#)^[142].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the connector to be created: <ul style="list-style-type: none"> <i>Type</i>: A string value corresponding to <i>Connector.Type</i> <i>Subtype</i>: A string value corresponding to <i>Connector.Subtype</i> <i>Stereotype</i>: A string value corresponding to <i>Connector.Stereotype</i> <i>ClientID</i>: A long value corresponding to <i>Connector.ClientID</i> <i>SupplierID</i>: A long value corresponding to <i>Connector.SupplierID</i> <i>DiagramID</i>: A long value corresponding to <i>Connector</i>.

Parameter	Type	Direction	Description
			<i>DiagramID.</i>
Repository	EA.Repository ^[197]	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.

6.5.7.3 EA_OnPreNewDiagram

Details

EA_OnPreNewDiagram notifies Add-Ins that a new diagram is about to be created. It enables Add-Ins to permit or deny creation of the new diagram.

The notification is provided immediately before the diagram is created, so that the Add-In can disable addition of the diagram.

Also look at [EA_OnPostNewDiagram](#)^[148].

Syntax

Function *EA_OnPreNewDiagram(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA_OnPreNewDiagram* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the diagram to be created: <ul style="list-style-type: none"> <i>Type</i>: A string value corresponding to <i>Diagram.Type</i> <i>ParentID</i>: A long value corresponding to <i>Diagram.ParentID</i> <i>PackageID</i>: A long value corresponding to <i>Diagram.PackageID</i>.
Repository	EA.Repository ^[197]	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 diagram to the model. Return **False** to disable addition of the new diagram.

6.5.7.4 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](#)^[148].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the object to be created: <ul style="list-style-type: none"> <i>Type</i>: A string value corresponding to <i>Object.Type</i> <i>Stereotype</i>: A string value corresponding to <i>Object.Stereotype</i> <i>ParentID</i>: A long value corresponding to <i>Object.ParentID</i> <i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the object is being added.
Repository	EA.Repository ^[197]	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.

6.5.7.5 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](#)^[149].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the attribute to be created: <ul style="list-style-type: none"> <i>Type</i>: A string value corresponding to <i>Attribute.Type</i> <i>Stereotype</i>: A string value corresponding to <i>Attribute.Stereotype</i> <i>ParentID</i>: A long value corresponding to <i>Attribute.ParentID</i> <i>ClassifierID</i>: A long value corresponding to <i>Attribute.ClassifierID</i>.
Repository	EA.Repository ^[197]	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.

6.5.7.6 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](#)^[149].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty Objects</i> for the method to be created: <ul style="list-style-type: none"> <i>ReturnType</i>: A string value corresponding to <i>Method.ReturnType</i> <i>Stereotype</i>: A string value corresponding to <i>Method.Stereotype</i> <i>ParentID</i>: A long value corresponding to <i>Method.ParentID</i> <i>ClassifierID</i>: A long value corresponding to <i>Method.ClassifierID</i>.
Repository	EA.Repository ^[197]	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.

6.5.7.7 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](#)^[150].

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	EA.EventProperties <small>[214]</small>	IN	Contains the following <i>EventProperty Objects</i> for the package to be created: <ul style="list-style-type: none"> <i>Stereotype</i>: A string value corresponding to <i>Package.Stereotype</i> <i>ParentID</i>: A long value corresponding to <i>Package.ParentID</i> <i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the package is being added.
Repository	EA.Repository <small>[197]</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 addition of the new package to the model. Return **False** to disable addition of the new package.

6.5.8 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	EA.Repository <small>[197]</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.

6.5.9 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](#)^[147]
- [EA_OnPostNewConnector](#)^[147]
- [EA_OnPostNewDiagram](#)^[148]
- [EA_OnPostNewDiagramObject](#)^[148]
- [EA_OnPostNewAttribute](#)^[149]
- [EA_OnPostNewMethod](#)^[149]
- [EA_OnPostNewPackage](#)^[150]

6.5.9.1 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](#)^[141].

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
Info	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new element: <ul style="list-style-type: none"> <i>ElementID</i>: A long value corresponding to <i>Element.ElementID</i>.
Repository	EA.Repository ^[197]	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.

6.5.9.2 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](#)^[142].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new connector: <ul style="list-style-type: none"> <i>ConnectorID</i>: A long value corresponding to <i>Connector.ConnectorID</i>.

Parameter	Type	Direction	Description
Repository	EA.Repository ^[197]	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.

6.5.9.3 EA_OnPostNewDiagram

Details

EA_OnPostNewDiagram notifies Add-Ins that a new diagram has been created. It enables Add-Ins to modify the diagram upon creation.

Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA_OnPreNewDiagram](#)^[143].

Syntax

Function *EA_OnPostNewDiagram*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA_OnPostNewDiagram* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new diagram: <ul style="list-style-type: none"> <i>DiagramID</i>: A long value corresponding to <i>Diagram.PackageID</i>.
Repository	EA.Repository ^[197]	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 diagram has been updated during this notification. Return **False** otherwise.

6.5.9.4 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](#)^[143].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new element: <ul style="list-style-type: none"> <i>ObjectID</i>: A long value corresponding to <i>Object.ObjectID</i>.
Repository	EA.Repository ^[197]	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.

6.5.9.5 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](#)^[144].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new attribute: <ul style="list-style-type: none"> <i>AttributeID</i>: A long value corresponding to <i>Attribute.AttributeID</i>.
Repository	EA.Repository ^[197]	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.

6.5.9.6 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](#)^[145].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new method: <ul style="list-style-type: none"> <i>MethodID</i>: A long value corresponding to <i>Method.MethodID</i>.
Repository	EA.Repository ^[197]	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.

6.5.9.7 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](#)^[145].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the new package: <ul style="list-style-type: none"> <i>PackageID</i>: A long value corresponding to <i>Package.PackageID</i>.
Repository	EA.Repository ^[197]	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.

6.5.10 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](#)^[20].

Syntax

Sub EA_OnPostInitialized(Repository As EA.Repository)

The *EA_OnPostInitialized* function syntax contains the following elements.

Parameter	Type	Direction	Description
Repository	EA.Repository ^[19]	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.

6.5.11 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	EA.EventProperties ^[21]	IN	Contains the following <i>EventProperty Objects</i> for the transform performed: <ul style="list-style-type: none"> <i>Transform</i>: A string value corresponding to the name of the transform used <i>PackageID</i>: A long value corresponding to <i>Package.PackageID</i> of the destination package.
Repository	EA.Repository ^[19]	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.

6.5.12 Technology Events

Enterprise Architect Add-Ins can respond to the following events associated with the use of MDG Technologies:

- [EA_OnInitializeTechnologies](#)^[152]
- [EA_OnPreActivateTechnology](#)^[152]
- [EA_OnPostActivateTechnology](#)^[153]
- [EA_OnPreDeleteTechnology](#)^[153] (Deprecated)
- [EA_OnDeleteTechnology](#)^[154] (Deprecated)
- [EA_OnImportTechnology](#)^[155] (Deprecated)

6.5.12.1 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	EA.Repository ^[197]	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
```

6.5.12.2 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 list box in the **Default Tools** toolbar - see *Using Enterprise Architect - UML Modeling Tool*).

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](#)^[153].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be activated: <ul style="list-style-type: none"> <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.
Repository	EA.Repository ^[197]	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.

6.5.12.3 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 list box in the **Default Tools** toolbar - see *Using Enterprise Architect - UML Modeling Tool*). 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](#)^[152].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be activated: <ul style="list-style-type: none"> <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.
Repository	EA.Repository ^[197]	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.

6.5.12.4 EA_OnPreDeleteTechnology

Deprecated - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA_OnPreActivateTechnology](#)^[152]
- [EA_OnPostActivateTechnology](#)^[153]
- [EA_OnInitializeTechnologies](#)^[152].

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](#)^[154].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be deleted: <ul style="list-style-type: none"> <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.
Repository	EA.Repository ^[197]	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.

6.5.12.5 EA_OnDeleteTechnology

Deprecated - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA_OnPreActivateTechnology](#)^[152]
- [EA_OnPostActivateTechnology](#)^[153]
- [EA_OnInitializeTechnologies](#)^[152].

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](#)^[153].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects: <ul style="list-style-type: none"> <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.
Repository	EA.Repository	IN	An <i>EA.Repository</i> object representing the currently open

Parameter	Type	Direction	Description
y	EA ^[197]		Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Return Value

None.

6.5.12.6 EA_OnImportTechnology

Deprecated - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA_OnPreActivateTechnology](#)^[152]
- [EA_OnPostActivateTechnology](#)^[153]
- [EA_OnInitializeTechnologies](#)^[152].

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	EA.EventProperties ^[214]	IN	Contains the following <i>EventProperty</i> objects: <ul style="list-style-type: none"> • <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.
Repository	EA.Repository ^[197]	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.

6.5.13 Context Item Events

Enterprise Architect Add-Ins can respond to the following events associated with changing context:

- [EA_OnContextItemChanged](#)^[155]
- [EA_OnContextItemDoubleClicked](#)^[156]
- [EA_OnNotifyContextItemModified](#)^[157]

6.5.13.1 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](#)^[135].

Also look at [EA_OnContextItemDoubleClicked](#)^[156] and [EA_OnNotifyContextItemModified](#)^[157].

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 ot 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> <i>otRepository</i> - NOT APPLICABLE, GUID is an empty string
ot	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
Repository	EA.Repository ^[197]	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.

6.5.13.2 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](#)^[155] and [EA_OnNotifyContextItemModified](#)^[157].

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
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 ot

Parameter	Type	Direction	Description
			parameter: <i>ot (ObjectType)</i> - GUID value <i>otElement</i> - Element.ElementGUID <i>otPackage</i> - Package.PackageGUID <i>otDiagram</i> - Diagram.DiagramGUID <i>otAttribute</i> - Attribute.AttributeGUID <i>otMethod</i> - Method.MethodGUID <i>otConnector</i> - Connector.ConnectorGUID
ot	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
Repository	EA.Repository ^[197]	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.

6.5.13.3 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](#)^[155] and [EA_OnContextItemDoubleClicked](#)^[156].

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 ot parameter: <ul style="list-style-type: none"> <i>ot (ObjectType)</i> - GUID value <i>otElement</i> - Element.ElementGUID <i>otPackage</i> - Package.PackageGUID <i>otDiagram</i> - Diagram.DiagramGUID <i>otAttribute</i> - Attribute.AttributeGUID <i>otMethod</i> - Method.MethodGUID <i>otConnector</i> - Connector.ConnectorGUID

Parameter	Type	Direction	Description
ot	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
Repository	EA.Repository <small>[197]</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.

6.5.14 Compartment Events

Enterprise Architect Add-Ins can respond to the following events associated with user-generated element compartments:

- [EA_QueryAvailableCompartments](#)
[158]
- [EA_GetCompartmentData](#)
[159]

6.5.14.1 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](#)
[159] 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	EA.Repository <small>[197]</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

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
```

End Function

6.5.14.2 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
oType	<i>ObjectType</i>	IN	The type of the element for which data is being requested.
Repository	EA.Repository <small>[197]</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.
sCompartment	<i>String</i>	IN	The name of the compartment for which data is being requested.
sGUID	<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

```

6.5.15 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](#)^[160] is intercepted in order to define rule categories and rules.
- [EA_OnStartValidation](#)^[161] 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](#)^[162]
 - [EA_OnRunPackageRule](#)^[162]
 - [EA_OnRunDiagramRule](#)^[162]
 - [EA_OnRunConnectorRule](#)^[163]
 - [EA_OnRunAttributeRule](#)^[163]
 - [EA_OnRunMethodRule](#)^[164]
- [EA_OnEndValidation](#)^[161] can be intercepted to perform any required clean-up after validation has completed.

Also consider the [Model Validation Example](#)^[165].

6.5.15.1 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](#)^[27] section.

Syntax

Sub EA_OnInitializeUserRules(Repository As EA.Repository)

The *EA_OnInitializeUserRules* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository <small>(197)</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.

6.5.15.2 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	EA.Repository <small>(197)</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.

6.5.15.3 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	EA.Repository <small>(197)</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.

6.5.15.4 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](#)^[160].

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	EA.Repository ^[197]	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.

6.5.15.5 EA_OnRunPackageRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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	EA.Repository ^[197]	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.

6.5.15.6 EA_OnRunDiagramRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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	Long	IN	The ID of the diagram to potentially perform validation on. Use the <i>Repository.GetDiagramByID</i> method to retrieve the diagram object.
Repository	EA.Repository [197]	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.

6.5.15.7 EA_OnRunConnectorRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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	EA.Repository [197]	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.

6.5.15.8 EA_OnRunAttributeRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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	EA.Repository [197]	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.

6.5.15.9 EA_OnRunMethodRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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
MethodGUID	String	IN	The GUID of the method to potentially perform validation on. Use the <i>Repository.GetMethodByGuid</i> method to retrieve the method object.
ObjectID	Long	IN	The ID of the object that owns the given method. Use the <i>Repository.GetObjectByID</i> method to retrieve the object.
Repository	EA.Repository [197]	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.

6.5.15.10 EA_OnRunParameterRule

Details

This event is triggered once for each rule defined in [EA_OnInitializeUserRules](#)^[160] 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
MethodGUID	String	IN	The GUID of the method that owns the given parameter. Use the <i>Repository.GetMethodByGuid</i> method to retrieve the method object.
ObjectID	Long	IN	The ID of the object that owns the given parameter. Use the <i>Repository.GetObjectByID</i> method to retrieve the object.
ParameterGUID	String	IN	The GUID of the parameter to potentially perform validation on. Use it to retrieve the parameter by iterating through the <i>Method.Parameters</i> collection.
Repository	EA.Repository [197]	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.

6.5.15.11 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.EnumMVErrorType.mvError,
GetRuleStr(cRule01)), cRule01);
        AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrorType.mvError,
GetRuleStr(cRule02)), cRule02);
        AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrorType.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));
    }
}
}
}

```

6.5.16 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](#)^[59] 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	EA.Repository ^[197]	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 <ModelTemplates> 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](#)^[169] 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](#)^[197] 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](#)^[171] event.

MDG Add-Ins identify themselves as such during [EA_Connect](#)^[130] 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](#)^[170].

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](#)^[171]
- [MDG_Connect](#)^[171]
- [MDG_Disconnect](#)^[172]

- [MDG_GetConnectedPackages](#)^[173]
- [MDG_GetProperty](#)^[173]
- [MDG_Merge](#)^[174]
- [MDG_NewClass](#)^[175]
- [MDG_PostGenerate](#)^[176]
- [MDG_PostMerge](#)^[176]
- [MDG_PreGenerate](#)^[177]
- [MDG_PreMerge](#)^[177]
- [MDG_PreReverse](#)^[178]
- [MDG_RunExe](#)^[179]
- [MDG_View](#)^[179]

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](#)^[179].

Syntax

Sub MDG_BuildProject(Repository As EA.Repository, PackageGuid As String)

The *MDG_BuildProject* function syntax contains the following elements:

Parameter	Type	Direction	Description
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package subtree that is controlled by the Add-In.
Repository	EA.Repository ^[197]	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](#)^[172].

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
PackageGuid	String	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.
PackageID	Long	IN	The <i>PackageID</i> of the Enterprise Architect package the user has requested to have connected to an external project.
Repository	EA.Repository ^[197]	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](#)^[171].

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
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
Repository	EA.Repository ^[197]	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](#)^[17].

Syntax

Function MDG_GetConnectedPackages(*Repository As EA.Repository*) *As Variant*

The *MDG_GetConnectedPackages* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	EA.Repository ^[19]	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	EA.Repository	IN	An <i>EA.Repository</i> object representing the currently-open Enterprise Architect model. Poll its members to retrieve model

Parameter	Type	Direction	Description
	197		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](#)^[177], [MDG_PreMerge](#)^[177] and [MDG_PostMerge](#)^[176].

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
ExportFiles	<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> .
ExportObjects	<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.
IgnoreLocked	<i>String</i>	OUT	A value indicating whether to ignore any files locked by the code project (that is, "TRUE" or "FALSE").
ImportFiles	<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.
Language	<i>String</i>	OUT	The string value containing the name of the code language supported by the code project connected to the model branch.
PackageGuid	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
Repository	EA.Repository ^[197]	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.
SynchObjects	<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 below ^[175] for the format of the Object IDs.
SynchType	<i>String</i>	OUT	The value determining the user-selected type of synchronization to take place. See below ^[175] 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](#)^[177].

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
CodeID	String	IN	A string used to identify the code element before it is created, for more information see MDG_View ^[178] .
Language	String	OUT	A string used to identify the programming language for the new Class. The language must be supported by Enterprise Architect.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
Repository	EA.Repository	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model

Parameter	Type	Direction	Description
	197		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](#)^[177].

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
FileContents	<i>String</i>	IN	A string containing the proposed contents of the file.
FilePath	<i>String</i>	IN	The path of the file Enterprise Architect intends to overwrite.
PackageGuid	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
Repository	EA.Repository ^[197]	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](#) ^[177], [MDG_PostGenerate](#) ^[176] and [MDG_PreReverse](#) ^[178].

Also look at [MDG_PreMerge](#) ^[177] and [MDG_Merge](#) ^[174].

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	EA.Repository ^[197]	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](#) ^[176].

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	EA.Repository ^[197]	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](#)^[177], [MDG_PostGenerate](#)^[176] and [MDG_PreReverse](#)^[178].

Also look at [MDG_Merge](#)^[174] and [MDG_PostMerge](#)^[176].

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	EA.Repository ^[197]	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](#)^[176] and [MDG_PreGenerate](#)^[177].

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	EA.Repository ^[197]	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](#)^[17].

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	EA.Repository ^[19]	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:

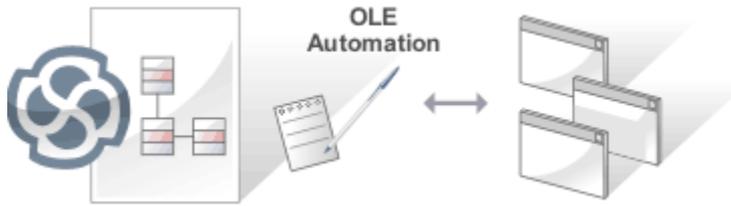
Parameter	Type	Direction	Description
Repository	<i>EA.Repository</i>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect 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

Parameter	Type	Direction	Description
			# - Class \$ - attribute % - operation For example if a user has selected the <i>m_Name</i> attribute of <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](#)^[187] 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](#)^[183]; 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](#)^[184] and other [available resources](#)^[186]. Also, consult the extensive [Reference Section](#)^[186].

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](#)^[185] topic. A more sophisticated method is to create [Add-Ins](#)^[123], 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](#)^[187]
- [Set References In Visual Basic](#)^[183]
- [Examples and Tips](#)^[184]

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](#)^[186] 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");
}
```

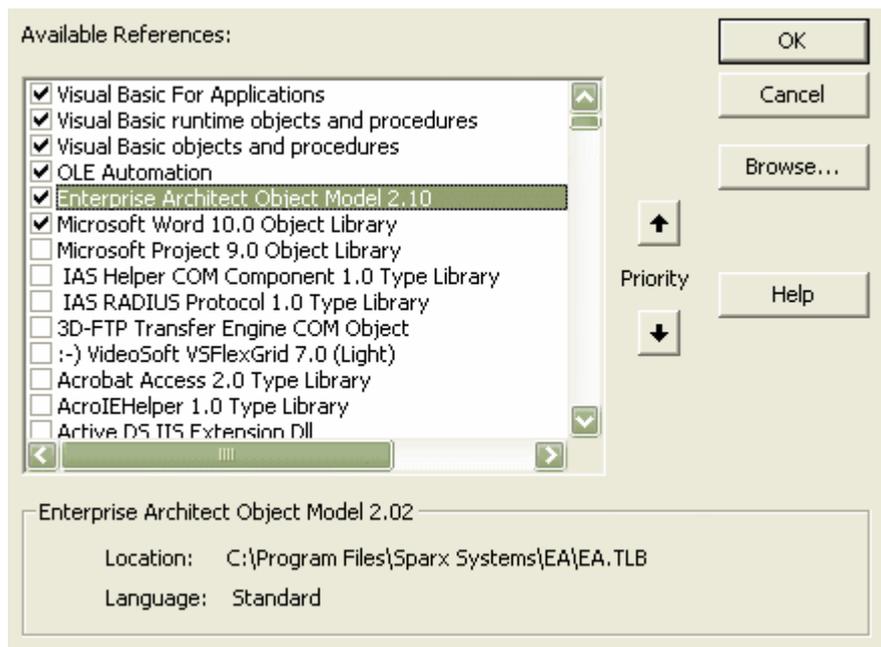
7.1.1.1 Set References In Visual Basic

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:

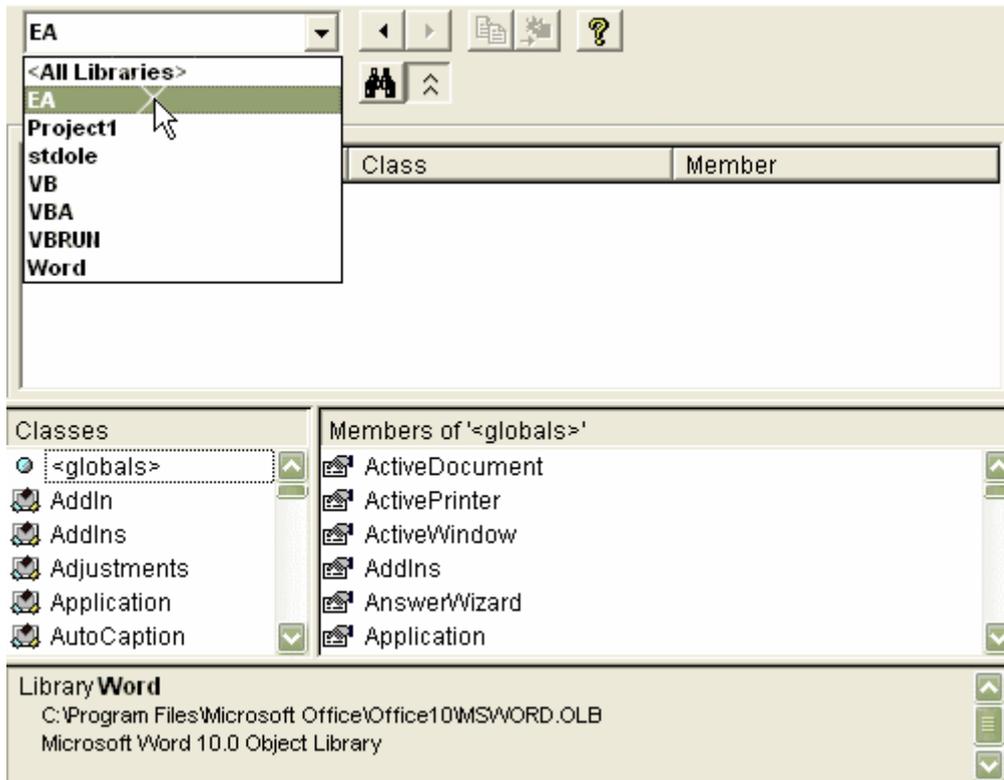
If Enterprise Architect Object Model 2.10 does not appear in the list, go to the command line and manually re-enter Enterprise Architect using the following:

- 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](#)^[185]
- Several VB.NET code snippets are provided in the [reference section](#)^[284]
- 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
- Additional samples are available from the Sparx Systems website; see the [Available Resources](#)^[186] 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](#)^[190] object. Accessing your Enterprise Architect model via the `App` object enables querying the current User Interface status, such as using `GetCurrentItem()` on the [Repository](#)^[197] 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:

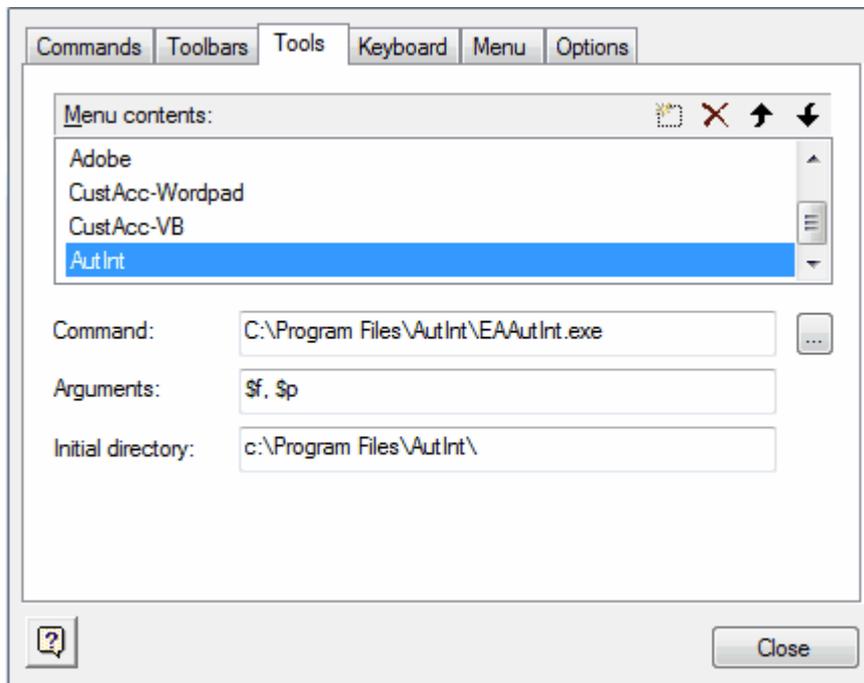
```
GC.Collect();
GC.WaitForPendingFinalizers();
```

There are additional concerns when controlling a running instance of Enterprise Architect that loads Add-Ins - see the [Tricks and Traps](#)^[126] 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.	www.sparxsystems.com/resources/developers/autint_vb.html
VB 6 Add-In to display a custom ActiveX graph control within the Enterprise Architect window as a new view.	www.sparxsystems.com/resources/developers/autint_vb_custom_view.html
A basic Add-In framework written in C#. Useful as a starting point for authoring your own custom Enterprise Architect Add-In.	www.sparxsystems.com/bin/CS_AddinFramework.zip
An extension on the <i>CS_AddinFramework</i> example showing how to export Tagged Values to a .csv file.	www.sparxsystems.com/bin/CS_AddinTaggedCSV.zip
A basic Add-In skeleton written in Delphi.	www.sparxsystems.com/bin/DelphiDemo.zip
A simple example Add-In written in C#.	www.sparxsystems.com/bin/CS_Sample.zip

For further information, see 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](#)^[187]
- [App](#)^[190]
- [Enumerations](#)^[191]
- [Repository](#)^[196]

- [Element](#) ^[225]
- [Element Features](#) ^[243]
- [Connector](#) ^[255]
- [Diagram Package](#) ^[263]
- [Project Interface](#) ^[271]
- [Code Samples](#) ^[284]

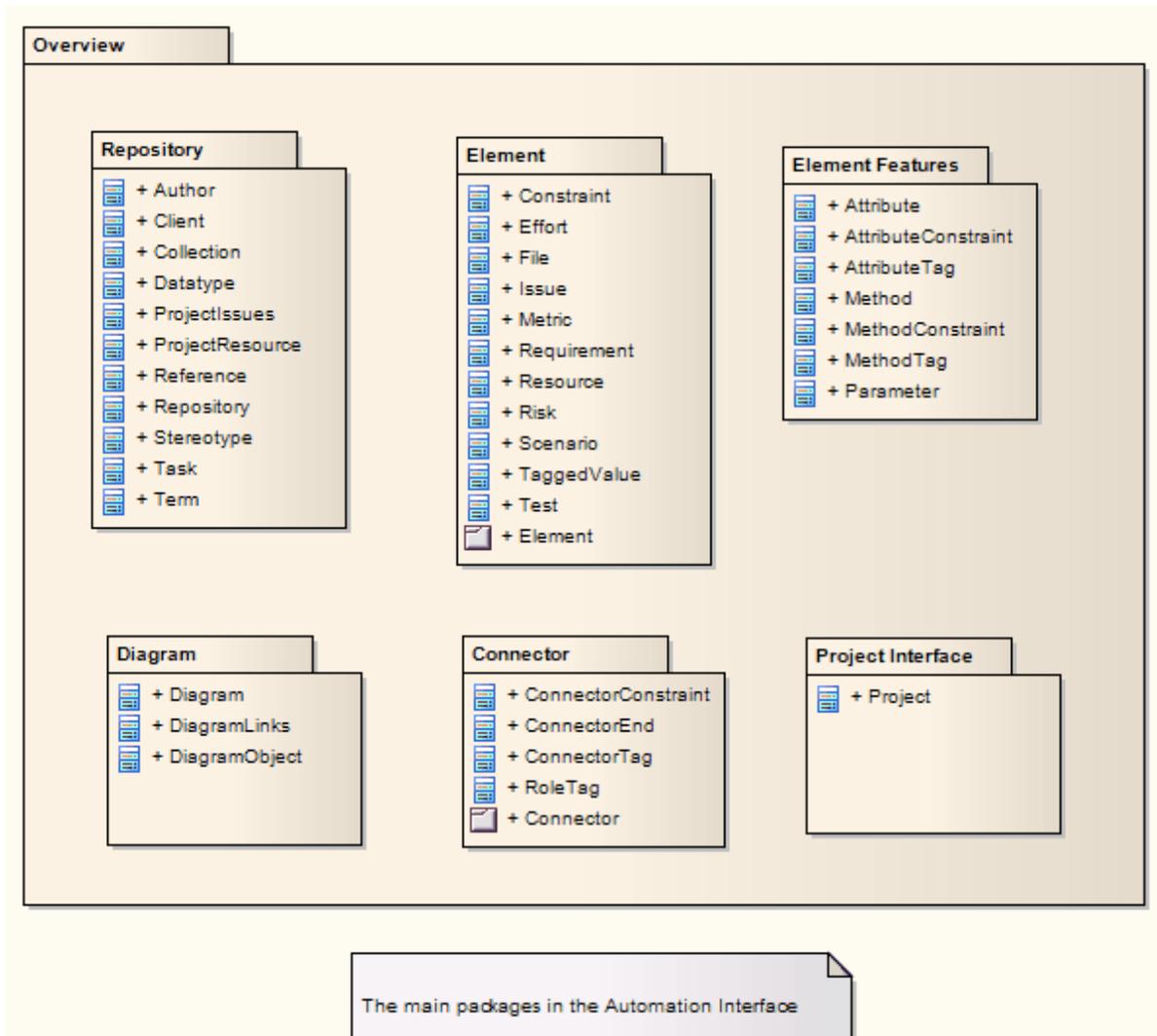
7.2.1 Interface Overview

public Package

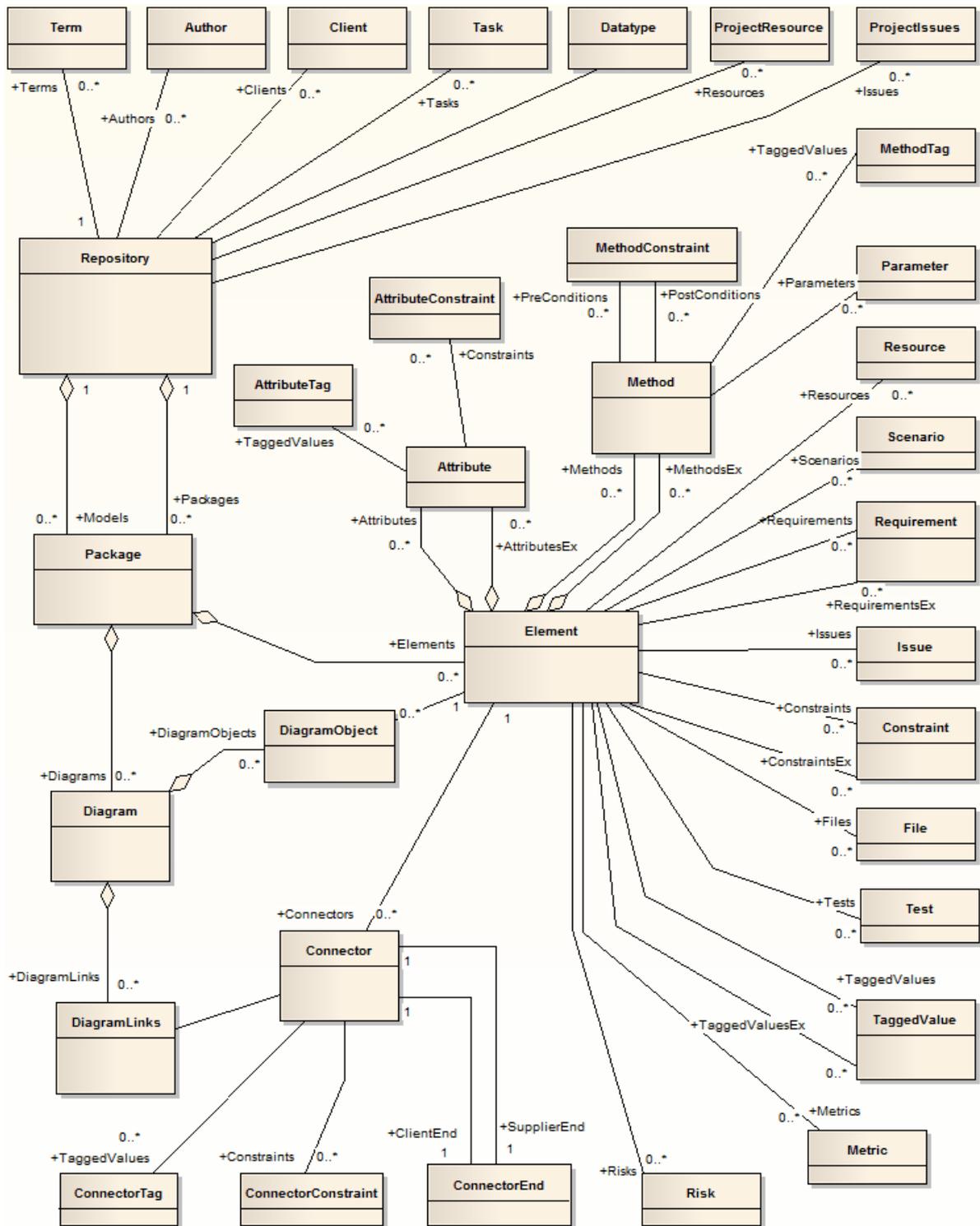
This package provides an overview of the main elements within the Automation Interface. These are:

- The [Repository](#) ^[196], which represents the model as a whole and provides entry to model packages and collections
- [Elements](#) ^[225], which are the basic structural unit (such as Class, Use Case and Object)
- [Element Features](#) ^[243], which are attributes and operations defined on an element
- [Diagram Package](#) ^[263], the visible drawings contained in the model
- [Connectors](#) ^[255], 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](#) ^[191]
- [CreateBaselineFlag](#) ^[192]
- [CreateModelType](#) ^[192]
- [EAEditionTypes](#) ^[192]
- [EnumRelationSetType](#) ^[192]
- [ExportPackageXMIFlag](#) ^[193]
- [MDGMenus](#) ^[193]
- [ObjectType](#) ^[193]
- [PropType](#) ^[194]
- [ReloadType](#) ^[194]
- [ScenarioDiagramType](#) ^[194]
- [ScenarioStepType](#) ^[195]
- [ScenarioTestType](#) ^[195]
- [XMIType](#) ^[195]

7.2.3.1 ConstLayoutStyles Enum

The *enum* values defined here are used exclusively for the *Lay Out 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 *Lay Out 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 CreateBaselineFlag Enum

The *CreateBaselineFlag* enumeration is used in Baseline Management, when creating a Baseline.

Method	Use to
<i>cbSaveToStub</i>	Baseline this package with only immediate children. (Child packages are included as stubs only.)

7.2.3.3 CreateModelType Enum

The *CreateModelType* enumeration is used in the [CreateModel](#)^[20†] 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.

7.2.3.4 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.

7.2.3.5 EnumRelationSetType Enum

This enumeration represents values returned from the *GetRelationSet* method of the [Element](#)^[228†] 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.

Method	Notes
<i>rsRealizeEnd</i>	List of elements that are realized by the current element.
<i>rsRealizeStart</i>	List of elements that the current element realizes.

7.2.3.6 ExportPackageXMIFlag Enum

The *ExportPackageXMIFlag* enumeration is used in package control, when exporting to XMI.

Method	Use to
<i>epSaveToStub</i>	Export this package with only immediate children. (Child packages are included as stubs only.)

7.2.3.7 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](#)^[173] topic for an example of use.

Method	Use to
<i>mgBuildProject</i>	Hide Build Project menu option.
<i>mgMerge</i>	Hide Merge menu option.
<i>mgRun</i>	Hide Run menu option.

7.2.3.8 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>otConnectorConstraint</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>

7.2.3.9 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.

7.2.3.10 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.

7.2.3.11 ScenarioDiagramType Enum

The *ScenarioDiagramType* enumeration provides the following enumeration values to the [Project.GenerateDiagramFromScenario\(\)](#) ^[27] method. They specify the type of diagram to generate. See the *Scenarios* section in *UML Modeling With Enterprise Architect - UML Modeling Tool*.

Method	Use to
<i>sdActivity</i>	Generate an Activity diagram.

Method	Use to
<i>sdActivityWithAction</i>	Generate an Activity diagram with an Action.
<i>sdActivityWithActionPin</i>	Generate an Activity diagram with an ActionPin.
<i>sdActivityWithActivityParameter</i>	Generate an Activity diagram with an ActivityParameter.
<i>sdRobustness</i>	Generate a Robustness diagram.
<i>sdRuleFlow</i>	Generate a RuleFlow diagram.
<i>sdSequence</i>	Generate a Sequence diagram.
<i>sdState</i>	Generate a State Machine diagram.

7.2.3.12 ScenarioStepType Enum

The *ScenarioStepType* enumeration is used to identify the [steps](#) ^[24↑] of a scenario, and the entity performing the step.

Method	Use to
<i>stSystem</i>	Identify that the step is an action performed by the system.
<i>stActor</i>	Identify that the step is an action performed by an actor.

7.2.3.13 ScenarioTestType Enum

The *ScenarioTestType* enumeration provides the following enumeration values to the [Project.GenerateTestFromScenario\(\)](#) ^[27↑] method. They specify the type of test to generate. See the *Scenarios* section in *UML Modeling With Enterprise Architect - UML Modeling Tool*.

Method	Use to
<i>stExternal</i>	Generate an external Test Case element.
<i>stInternal</i>	Generate an internal test.

7.2.3.14 XMISType Enum

The following enumeration values are used in the *Project.ExportPackageXML()* method. They enable specification of the XML export type.

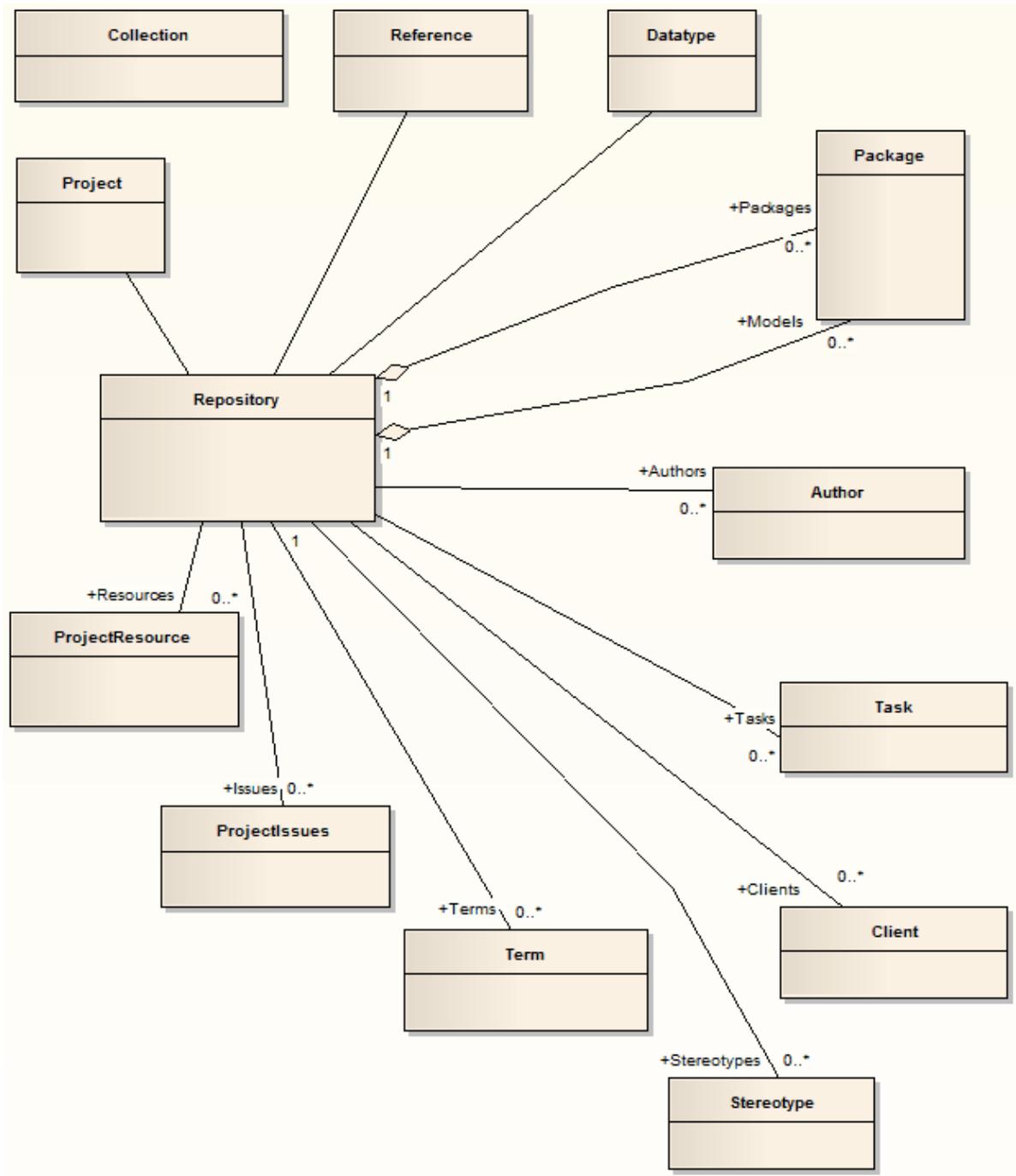
xmiEADefault
xmiRoseDefault
xmiEA10
xmiEA11
xmiEA12
xmiRose10
xmiRose11
xmiRose12
xmiMOF13
xmiMOF14
xmiEA20
xmiEA21

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](#)^[197] 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
Authors	Collection ^[21]	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.
BatchAppend	<i>Boolean</i>	Read/Write. Set this property to true when your automation client has to rapidly insert many elements, operations, attributes and/or operation parameters. Set to false when work is complete. This can result in 10- to 20-fold improvement in adding new elements in bulk.
Clients	Collection ^[21]	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.
ConnectionString	<i>String</i>	Read only. The filename/connection string of the current Repository.
Datatypes	Collection ^[21]	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.
EAEdition	EAEditionType ^[192]	Read only. Returns the level of core licensed functionality available to the current repository. Note: This property returns Corporate when the edition is <i>Business and Software Engineering, Systems Engineering</i> or <i>Ultimate</i> . Use EAEditionEx to identify which of these extended editions is available.
EAEditionEx	EAEditionType ^[192]	Read only. Returns the level of extended licensed functionality available to the current repository.
EnableCache	<i>Boolean</i>	Read/Write. An optimization for pre-loading package objects when dealing with large sets of automation objects.
EnableUIUpdates	<i>Boolean</i>	Read/Write. Set this property to false 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.RefreshModelView()</i> .
FlagUpdate	<i>Boolean</i>	Read/Write. Instructs Enterprise Architect to update the Repository with the <i>LastUpdate</i> value.
InstanceGUID	<i>String</i>	Read only. The identifier string identifying the Enterprise Architect runtime session.

Attribute	Type	Notes
IsSecurityEnabled	Boolean	Read only. Checks whether User Security is enabled for the current repository.
Issues	Collection ^[21†]	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.
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	Collection ^[21†] of type Package ^[215]	<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 Project Browser 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 0. 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	ObjectType ^[193]	Read only. Distinguishes objects referenced through the Dispatch interface.
ProjectGUID	String	Read only. Returns a unique ID for the project.
PropertyTypes	Collection ^[21†]	Read only. Collection of Property Types ^[22†] available to the Repository.
Resources	Collection ^[21†]	Read only. Contains available <i>ProjectResource</i> objects to assign to work items within the project. Use the add new , modify and delete functions to manage resources.
Stereotypes	Collection ^[21†]	Read only. The Stereotypes ^[223] 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 EA_OnPostNewElement ^[147] or EA_OnPostNewConnector ^[147] broadcast events to control whether Enterprise Architect should suppress showing the default Properties dialogs to the user when an element or connector is created.
Tasks	Collection ^[21†]	Read only. A list of system tasks (to do list). Each entry is a Task ^[223] <i>Item</i> ; you can modify, delete and add new tasks.
Terms	Collection ^[21†]	Read only. The project <i>Glossary</i> . Each Term ^[224] object is an entry in the Glossary. Add, modify and delete Terms to maintain the Glossary.

Repository Methods

Method	Type	Notes
ActivateDiagram (long DiagramID)		Activates an already open diagram (that is, makes it the active tab) in the main Enterprise Architect user interface. Parameters: <ul style="list-style-type: none"> DiagramID: Long - the ID of the diagram to make active.
ActivatePerspective (string, long)	<i>Boolean</i>	Deprecated - no longer in use.
ActivateTab (string Name)		Activates an open Enterprise Architect tabbed view. Parameters: <ul style="list-style-type: none"> Name: String - the name of the view to activate.
ActivateTechnology (string Name)		Activates an enabled MDG Technology. Parameters: <ul style="list-style-type: none"> Name: String - the name of the Technology to activate.
ActivateToolbox (string Toolbox, long Options)	<i>Boolean</i>	Activates a Toolbox page in the GUI. Returns true if the specified Toolbox page is successfully activated, otherwise returns false . Parameters: <ul style="list-style-type: none"> Toolbox: String - the name of the Toolbox page to activate. Options: Long - reserved for future use.
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"> sXML: String - the XML of the defined searches; you can get this XML by performing an <i>export</i> of the searches from the Manage Searches dialog in Enterprise Architect. (See <i>Using Enterprise Architect - UML Modeling Tool</i>.)
AddPerspective (string Perspective, long Options)	<i>Boolean</i>	Deprecated - no longer in use.
AddTab (string TabName, string ControlID)	<i>activeX custom control</i>	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"> TabName: String - used as the tab caption. ControlID: String - the ProgID of the control; for example, Project1, UserControl1.
AddWindow (string WindowName, string ControlID)	<i>activeX custom control</i>	Adds an ActiveX custom control as a window to the Add-Ins docked window. Enterprise Architect creates a control and, if successful, returns its Unknown pointer, which can be used by the caller to manipulate the control. The window can be shown by selecting it from the list in the Workspace Layouts toolbar (see <i>Using Enterprise Architect</i>

Method	Type	Notes
		<p>- <i>UML Modeling Tool</i>) - click on the third icon from the <i>right</i> and look at the end of the list.</p> <p>Parameters:</p> <ul style="list-style-type: none"> WindowName: String - used as the window title. ControlID: String - the ProgID of the control; for example, Project1,UserControl1.
AdviseConnectorChange (long ConnectorID)		<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> ConnectorID: Long - the ID of the connector.
AdviseElementChange (long ObjectID)		<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> ObjectID: Long - the ID of the element.
ChangeLoginUser (string Name, string Password)	Boolean	<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Name: String - the name of the user. Password: String - the password of the user.
ClearAuditLogs (Object StartDateTime, Object EndDateTime)	Boolean	<p>Clears all Audit Logs from the model.</p> <p>If <i>StartDateTime</i> and <i>EndDateTime</i> are not null then only log items that fall into this period are cleared.</p> <p>Returns true for success, false for failure.</p> <p>Notes:</p> <ul style="list-style-type: none"> This method cannot be undone. It is strongly advised that you call <i>SaveAuditLogs</i> first to backup the logs. This method might fail if the user logged into the model does not have the correct access permission. <p>Parameters:</p> <ul style="list-style-type: none"> StartDateTime: Variant [DateTime] - the earliest date and time of log entries to clear. EndDateTime; Variant [DateTime] - the latest date and time of log entries to clear.
ClearOutput (string Name)		<p>Removes all the text from a tab in the Output window. See also CreateOutputTab^[20†], EnsureOutput Visible^[20†], WriteOutput^[20†].</p> <p>Parameters:</p> <ul style="list-style-type: none"> Name: String - the name of the tab to remove text from.
CloseAddins ()		Called by automation controllers to ensure that Add-Ins

Method	Type	Notes
		created in .NET do not linger after all controller references to Enterprise Architect have been cleared.
CloseDiagram (long DiagramID)		Closes a diagram in the current list of diagrams that Enterprise Architect has open. Parameters: <ul style="list-style-type: none"> DiagramID: Long - the ID of the diagram to close.
CloseFile ()		Closes any open file.
CreateModel (CreateModelType CreateType, string FilePath, long ParentWnd)	<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 true when the new file is created, otherwise returns false . Parameters: <ul style="list-style-type: none"> CreateType: CreateModelType^[192] - Specify whether to make a new copy of the EABase.eap model, or 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. FilePath: String - Destination for new .eap file. ParentWnd: Long - Window handle to act as the parent for the SQL connection dialog. Only required when using <i>cmEAPFromSQLRepository</i>.
CreateOutputTab (string Name)		Creates a tab in the Output window. See also ClearOutput ^[200] , EnsureOutput Visible ^[201] , WriteOutput ^[209] . Parameters: <ul style="list-style-type: none"> Name: String - the name of the tab to create.
DeletePerspective (string Perspective, long Options)	<i>Boolean</i>	Deprecated - no longer in use.
DeleteTechnology (string ID)	<i>Boolean</i>	Removes a specified MDG Technology resource from the repository. Returns true , if the technology is successfully removed from the model. Returns false otherwise. Note: 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 Deploying MDG Technologies ^[60] (from Add-Ins). Parameters: <ul style="list-style-type: none"> ID: String - the ID of the technology.
EnsureOutputVisible (string Name)		Ensures that a specified tab in the Output window is visible to the user. The Output window is made visible if it is hidden. See also ClearOutput ^[200] , CreateOutputTab ^[201] , WriteOutput ^[209] . Parameters: <ul style="list-style-type: none"> Name: String - the name of the tab to make visible.

Method	Type	Notes
ExecutePackageBuildScript (long ScriptOptions, string PackageGuid)		Enables you to run the active package build script based on your current selection in the Project Browser . You can also run a script by passing in the package GUID. Parameters: <ul style="list-style-type: none"> ScriptOptions: Long - the script type; can be any one of these numerical values: <ul style="list-style-type: none"> 1 = Build 2 = Test 3 = Run 4 = Create Workbench Instance 5 = Debug. PackageGuid: String - the ID of the package for which to run the script.
Exit		Shuts down Enterprise Architect immediately. Used by .NET programmers where the garbage collector does not immediately release all referenced COM objects.
GetActivePerspective ()	String	Deprecated - no longer in use.
GetAttributeByGuid (string Guid)	Attribute [244]	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"> Guid: String - the GUID of the attribute to locate.
GetAttributeByID (string Id)	Attribute [244]	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"> Id: String - the ID of the attribute to locate.
GetConnectorByGuid (string Guid)	Connector [257]	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"> Guid: String - the GUID of the connector to locate.
GetConnectorByID (long ConnectorID)	Connector [257]	Searches the repository for a connector with a specific ID. Parameters: <ul style="list-style-type: none"> ConnectorID: Long - the ID of the connector to locate.
GetContextItem (object Item)	ObjectType e [193]	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 GetContextItemType method (below). Parameters: <ul style="list-style-type: none"> Item: Object - the item to point to.
GetContextItemType ()	ObjectType e [193]	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:

Method	Type	Notes
		<ul style="list-style-type: none"> An item selected in the Project Browser An item selected in an open diagram An item selected in certain dialogs, such as the attribute Properties dialog. <p>The supported <i>ObjectTypes</i> can be any one of the following values:</p> <ul style="list-style-type: none"> <i>otElement</i> <i>otPackage</i> <i>otDiagram</i> <i>otAttribute</i> <i>otMethod</i> <i>otConnector</i>
GetCurrentContextObject ()	<i>Object</i>	Returns the current context Object.
GetCounts ()	<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.
GetCurrentDiagram ()	Diagram <small>[264]</small>	Returns a selected diagram.
GetCurrentLoginUser (boolean GetGuid = false)	<i>String</i>	<p>If security is not enabled in the repository, an error is generated.</p> <p>If <i>GetGuid</i> is True, 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.</p>
GetDiagramByGuid (string Guid)	Diagram <small>[264]</small>	<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Guid: String - the GUID of the diagram to locate.
GetDiagramByID (long DiagramID)	Diagram <small>[264]</small>	<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> DiagramID: Long - the ID of the diagram to locate.
GetElementByGuid (string Guid)	Element <small>[228]</small>	<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Guid: String - the GUID of the element to locate.
GetElementByID (long ElementID)	Element <small>[228]</small>	<p>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.</p> <p>Parameters:</p>

Method	Type	Notes
		<ul style="list-style-type: none"> ElementID: Long - the ID of the element to locate.
GetElementsByQuery (string QueryName, string SearchTerm)		<p>Enables the user to run a search in Enterprise Architect, returning the result as a collection.</p> <p>For example <i>GetElementsByQuery('Simple','Class1')</i>, where results contain elements with <i>Class1</i> in the Name and Notes fields.</p> <p>Parameters:</p> <ul style="list-style-type: none"> QueryName: String - the name of the search to run, for example 'Simple'. SearchTerm: String - the term to search for.
GetElementSet (string IDList, long Options)	Collection ^[211]	<p>Returns a set of elements as a collection based on a 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"> IDList: String - a comma-separated list of <i>ElementID</i> values Options: Long - modifies default behaviour of this method <ul style="list-style-type: none"> 1 - Returns empty collection when empty IDList parameter is given. 2 - Use IDList string as an SQL query to populate this collection.
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"> Format: String - The format to convert the field from. Valid formats are: <ul style="list-style-type: none"> HTML - Full HTML RTF - Rich Text Format TXT - Plain text Text: String - The field to be converted.
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"> Format: String - The format to convert the field to. Valid formats are: <ul style="list-style-type: none"> HTML - Full HTML RTF - Rich Text Format TXT - Plain text Text: String - The field to be converted.
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)	Method ^[249]	<p>Returns a pointer to a method in the repository. This is usually found using the <i>MethodGUID</i> property of a method.</p>

Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> Guid: String - the GUID of the method to look for.
GetMethodById (string Id)	Method ^[249]	Returns a pointer to a method in the repository. This is usually found using the <i>MethodID</i> property of a method. Parameters: <ul style="list-style-type: none"> Id: String - the ID of the method to look for.
GetPackageByGuid (string Guid)	Package ^[215]	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. 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: <i>GetPackageByGuid(Element.ElementGUID)</i> . Parameters: <ul style="list-style-type: none"> Guid: String - the GUID of the package to look for.
GetPackageById (long PackageID)	Package ^[215]	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. Parameters: <ul style="list-style-type: none"> PackageID: Long - the ID of the package to locate.
GetProjectInterface ()	Project ^[27]	Return a pointer to the EA.Project interface ^[27] (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.
GetReferenceList (string Type)	Reference ^[222]	Uses the list type to get a pointer to a <i>Reference List</i> object. Parameters: <ul style="list-style-type: none"> Type: String - specifies the list type to get; valid list types are: <ul style="list-style-type: none"> <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>.
GetTechnologyVersion (string ID)	<i>String</i>	Returns the version of a specified MDG Technology resource. Parameters: <ul style="list-style-type: none"> ID: String - the specified technology ID.
GetTreeSelectedElements()	Collection ^[21]	Returns the set of elements currently selected in the Project Browser as a collection.

Method	Type	Notes
GetTreeSelectedItem (object SelectedItem)	Object ^[193]	Gets an object variable and type corresponding to the currently selected item in the tree view. 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. The object passed back through the parameter can be a package, element, diagram, attribute or operation object. Parameters: <ul style="list-style-type: none"> SelectedItem: Object - the object to get the variable and type for.
GetTreeSelectedItemType ()	Object ^[193]	Returns the type of the object currently selected in the tree. One of: <ul style="list-style-type: none"> <i>otDiagram</i> <i>otElement</i> <i>otPackage</i> <i>otAttribute</i> <i>otMethod</i>.
GetTreeSelectedObject ()	<i>Object</i>	The related method GetTreeSelectedItem ^[206] () has an output parameter that is inaccessible by some scripting languages. As an alternative, this method provides the selected item through the return value.
GetTreeSelectedPackage ()	Package ^[215]	Returns the package in which the currently selected tree view object is contained.
HasPerspective (string Perspective)	<i>String</i>	Deprecated - no longer in use.
ImportPackageBuildScripts (string PackageGuid, string BuildScriptXML)		Imports build scripts into a package in Enterprise Architect. Parameters: <ul style="list-style-type: none"> PackageGuid: String - the GUID of the package into which to import the build scripts. BuildScriptXML: String - the build script XML data, which you can export from within Enterprise Architect.
ImportTechnology (string Technology)	<i>Boolean</i>	Installs a given MDG Technology resource into the repository. Returns True , if the technology is successfully loaded into the model. Otherwise returns False . Note: 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 Deploying MDG Technologies ^[60] (from Add-Ins). Parameters: <ul style="list-style-type: none"> Technology: String - the contents of the technology resource file.
IsTabOpen (string TabName)	<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 Repository.AddTab() ^[199] .

Method	Type	Notes
		<p>Returns:</p> <ul style="list-style-type: none"> • 2 to indicate that a tab is open and active (top-most) • 1 to indicate that it is open but not top-most, or • 0 to indicate that it is not visible at all. <p>Note:</p> <p>TabName is case-sensitive.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • TabName: String - the name of the tab to check for.
IsTechnologyEnabled (string ID)	<i>Boolean</i>	<p>Checks whether a specified technology is enabled in Enterprise Architect.</p> <p>Returns True if the MDG Technology resource is enabled. Otherwise returns False.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ID: String - the technology ID to check for.
IsTechnologyLoaded (string ID)	<i>Boolean</i>	<p>Checks whether a specified technology is loaded into the repository.</p> <p>Returns True if the MDG Technology resource is loaded into the repository. Otherwise returns False.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ID: String - the technology ID to check for.
OpenDiagram (long DiagramID)		<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • DiagramID: Long - the ID of the diagram to open.
OpenFile (string Filename)	<i>Boolean</i>	<p>This is the main point for opening an Enterprise Architect project file from an automation client, and working with the contained objects.</p> <p>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.</p> <p>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 Open Project dialog (see <i>Using Enterprise Architect - UML Modeling Tool</i>) by selecting a recently opened SQL repository.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Filename: String - the filename of the Enterprise Architect project to open.
OpenFile2 (string FilePath, string Username, string Password)	<i>Boolean</i>	<p>As for <i>OpenFile()</i> except this enables the specification of a password.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • FilePath: String - the file path of the Enterprise Architect project to open. • Username: String - the user login ID • Password: String - the user password.

Method	Type	Notes
RefreshModelView (long PackageID)		<p>Reloads a package or the entire model, updating the user interface.</p> <p>Parameters:</p> <ul style="list-style-type: none"> PackageID: Long - the ID of the package to reload: if 0, the entire model is reloaded; if a valid package ID, only that package is reloaded.
RefreshOpenDiagrams (boolean FullReload)		<p>Refreshes the diagram contents for all diagrams open in Enterprise Architect.</p> <p>Parameters:</p> <ul style="list-style-type: none"> FullReload: Boolean - if false the displayed contents of elements and connectors are refreshed in each diagram; if true each of the diagrams is completely reloaded from the repository.
ReloadDiagram (long DiagramID)		<p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> DiagramID: Long - the ID of the diagram to be reloaded.
RemoveOutputTab (string Name)		<p>Removes a specified tab from the Output window.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Name: String - the name of the tab to be removed.
RunModelSearch (string sQueryName, string sSearchTerm, string sSearchOptions, string sSearchData)		<p>Runs a search, displaying the results in Enterprise Architect's Model Search window.</p> <p>Parameters:</p> <ul style="list-style-type: none"> sQueryName: String - the name of the search to run, for example <i>Simple</i>. sSearchTerm: String - the term to search for. sSearchOptions: String - currently not being used. 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 XML Format^[129]; this parameter is not mandatory so pass in an empty string to run the search as per normal.
SaveAllDiagrams ()		<p>Saves all open diagrams.</p>
SaveAuditLogs (string FilePath, object StartDateTime, object EndDateTime)	<i>Boolean</i>	<p>Saves the Audit Logs contained within a model to a specified file.</p> <p>If <i>StartDateTime</i> and <i>EndDateTime</i> are not null then only log items that fall into this period are saved.</p> <p>Returns true for success, false for failure.</p> <p>Note:</p> <p>This might fail if the user logged into the model does not have the correct access permission.</p> <p>Parameters:</p> <ul style="list-style-type: none"> FilePath: String - the file to save the Audit Logs to. StartDateTime: Variant [DateTime] - the earliest date and time of log entries to save.

Method	Type	Notes
		<ul style="list-style-type: none"> EndDateTime; Variant [DateTime] - the latest date and time of log entries to save.
SaveDiagram (long DiagramID)		<p>Saves an open diagram. Assumes the diagram is open in the main user interface Tab list.</p> <p>Parameters:</p> <ul style="list-style-type: none"> DiagramID: Long - the ID of the diagram to save.
ShowDynamicHelp (string Topic)		<p>Shows a help topic as a view.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Topic: String - specifies the help topic.
ShowInProjectView (object Item)		<p>Selects a specified object in the Project Browser.</p> <p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Item: Object - the object to highlight.
ShowProfileToolbox (string Technology, string Profile, boolean Show)		<p>Shows/hides the contents of a specified technology or profile in the Enterprise Architect UML Toolbox.</p> <p>To show/hide a profile in the Toolbox, specify the profile's ID value in the <i>Profile</i> parameter and set the <i>Technology</i> parameter to a null string.</p> <p>To show/hide a technology in the Toolbox, specify the technology's ID in the <i>Technology</i> parameter and set the <i>Profile</i> parameter to a null string.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Technology: String - the ID of the technology. Profile: String - the ID of the profile. Show: Boolean - if true, show the technology or profile; if false, hide the technology or profile.
ShowWindow (long Show)		<p>Shows or hides Enterprise Architect.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Show: Long.
SQLQuery (string SQL)	String	<p>Enables execution of a SQL <i>select</i> statement against the current repository. Returns an XML formatted string value of the resulting recordset.</p> <p>Parameters:</p> <ul style="list-style-type: none"> SQL: String - contains the SQL Select statement.
VersionControlResynchPkgStatuses (boolean ClearSettings)		<p>Synchronizes the version control status of each version controlled package within the current model with the status reported by your version control provider. (See <i>Version Control Within UML Models Using Enterprise Architect</i>.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> ClearSettings: Boolean - if true, clear the version control settings from packages that are reported by the version control provider as <i>uncontrolled</i>; if false, leave the version control settings unchanged for packages reported as <i>uncontrolled</i>.
WriteOutput (string Name,		Writes text to a specified tab in the Output window, and

Method	Type	Notes
string String, long ID)		associates the text with an ID. See also ClearOutput ^[200] , CreateOutputTab ^[201] , EnsureOutput Visible ^[201] . Parameters: <ul style="list-style-type: none"> • Name: String - specifies the tab on which to display the text. • String: String - specifies the text to display. • ID: Long - specifies the ID the text is associated with.

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	ObjectType ^[193]	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 false 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	<i>String</i>	Read/Write. EMail address.
Fax	<i>String</i>	Read/Write. Fax number.
Mobile	<i>String</i>	Read/Write. Mobile phone if available.
Name	<i>String</i>	Read/Write. Client name.
Notes	<i>String</i>	Read/Write. Notes about client.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through the <i>Dispatch</i> interface.
Organization	<i>String</i>	Read/Write. Associated organization.
Phone1	<i>String</i>	Read/Write. Main phone number.
Phone2	<i>String</i>	Read/Write. Second phone number.
Roles	<i>String</i>	Read/Write. Roles this client might play in the project.

Client Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Updates the current Client object after modification or appending a new item. If false 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
Count	Short	Read only. The number of objects referenced by this list.
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.

Collection Methods

Method	Type	Notes
AddNew (string Name, string Type)	Object	<p>Adds a new item to the current collection.</p> <p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Name: String Type: String (up to 30 characters long)
Delete (short index)	Void	<p>Deletes the item at the selected reference.</p> <p>Parameters:</p> <ul style="list-style-type: none"> index: Short
DeleteAt (short index, boolean Refresh)	Void	<p>Deletes the item at the selected index. The second parameter is currently unused.</p> <p>Parameters:</p> <ul style="list-style-type: none"> index: Short Refresh: Boolean
GetAt (short index)	Object	<p>Retrieves the array object using a numerical index. If the index is out of bounds, an error occurs.</p> <p>Parameters:</p> <ul style="list-style-type: none"> index: Short
GetByName (string Name)	Object	<p>Gets an item in the current collection by Name.</p> <p>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.</p> <p>Only supported for the following collections: Models, Packages [215], Elements [228], Diagrams [264], and element TaggedValues [242].</p> <p>Parameters:</p> <ul style="list-style-type: none"> Name: String
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>
Refresh ()	Void	<p>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.</p>

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
DatatypeID	<i>Long</i>	Read/Write. Instance ID for this datatype within the current model. System maintained.
DefaultLen	<i>Long</i>	Read/Write. Default length (DDL only).
DefaultPrec	<i>Long</i>	Read/Write. Default precision (DDL only).
DefaultScale	<i>Long</i>	Read/Write. Default scale (DDL only).
GenericType	<i>String</i>	Read/Write. The associated generic type for this data type.
HasLength	<i>String</i>	Read/Write. Indicates datatype has a length component.
MaxLen	<i>Long</i>	Read/Write. Maximum length (DDL only).
MaxPrec	<i>Long</i>	Read/Write. Maximum precision (DDL only).
MaxScale	<i>Long</i>	Read/Write. Maximum scale (DDL only).
Name	<i>String</i>	Read/Write. The datatype name (such as <i>integer</i>). This appears in the related drop-down datatype lists where appropriate.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Product	<i>String</i>	Read/Write. The datatype product, such as Java, C++, Oracle.
Size	<i>Long</i>	Read/Write. The datatype size.
Type	<i>String</i>	Read/Write. The type can be <i>DDL</i> for database datatype or <i>Code</i> for language datatypes.
UserDefined	<i>Long</i>	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 Datatype dialog are marked 1 (true) .

Datatype Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Updates the current Datatype object after modification or appending a new item.

Method	Type	Notes
		If false 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	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.

EventProperties Methods

Method	Type	Notes
Get (object Index)	EventProperty ^[214]	Read only. Returns an <i>EventProperty</i> in the list, raising an error if <i>Index</i> is out of range. Parameters: <ul style="list-style-type: none"> 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>.

7.2.4.7 EventProperty

EventProperty objects are always part of an [EventProperties](#)^[214] collection, and are passed to Add-In methods responding to [broadcast events](#)^[135].

EventProperty Attributes

Attribute	Type	Notes
Description	String	Explanation of what this property represents.
Name	String	A string distinguishing this property from others in the list.
ObjectType	ObjectType ^[193]	Distinguishes objects referenced through a Dispatch interface.
Value	Variant	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	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.

ModelWatcher Methods

Methods	Type	Notes
GetReloadItem (object Item)	ReloadType [194]	<p>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 null (C#) or Nothing (VB).</p> <p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> Item: Object
PeekReloadItem	ReloadType [194]	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 XML export list or not.
CodePath	String	Read/Write. The path to where associated source code is found. Not currently used.
Connectors	Collection [211]	Read only. Collection of connectors.
Created	Date	Read/Write. Date the package was created.

Attribute	Type	Notes
Diagrams	Collection [21↑]	Read only. A collection of diagrams contained in this package.
Element	Element ^[228]	Read only. The associated element object. Use to get/set common information such as Stereotype, Complexity, Alias, Author, Constraints, Tagged Values and Scenarios.
Elements	Collection [21↑]	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. True is 'package is a Namespace root'. Use 0 and 1 to set False and True .
IsProtected	Boolean	Read/Write. Indicates if the package has been marked as <i>Protected</i> .
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 XML 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	ObjectType [193]	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	Collection [21↑]	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. 0 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 XML export purposes.
UseDTD	Boolean	Read/Write. Indicates if a DTD is to be used when exporting XML.

Attribute	Type	Notes
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"> aGroupName: String - The name of the security group for which to apply the lock.
ApplyUserLock ()	Boolean	Applies a user lock to the package object for the current user. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
Clone	LDISPATCH	Inserts a copy of the package into the same parent as the original package. Returns the newly-created package.
FindObject (string DottedID)	LPDISPATCH	Returns a package, element, attribute or operation matching the parameter <i>DottedID</i> . If the DottedID is not found, an error is returned: <i>Can't find matching object</i> . Parameter: <ul style="list-style-type: none"> 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.DoubleIt(int)</i>.
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.
ReleaseUserLock ()	Boolean	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.
Update ()	Boolean	Update the current package object after modification or appending a new item. If false 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.
VersionControlAdd (string	Void	Places the package under version control, using the specified Version Control Configuration and the specified XML filename.

Method	Type	Notes
ConfigGuid, string XMLFile, string Comment, boolean KeepCheckedOut)		<p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p> <p>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.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ConfigGuid: String - Name corresponding to the Unique ID of the version control configuration to use. • 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. • Comment: String - Log message that is added to the version controlled file's history (where applicable). • KeepCheckedOut: Boolean - Specify True to add to version control and keep package checked-out.
VersionControlCheckin (string Comment)	<i>Void</i>	<p>Perform checkin of the version controlled package.</p> <p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Comment: String - Log message that is added to the version controlled file's history (where applicable).
VersionControlCheckout (string Comment)	<i>Void</i>	<p>Perform checkout of the version controlled package.</p> <p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Comment: String - Log message that is added to the version controlled file's history (where applicable).
VersionControlGetLatest (boolean Forcelmport)		<p>Updates the local working copy of the package file associated with the object package, before re-importing the package data from the package file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Forcelmport: Boolean - Used if the package data in the model is found to be up-to-date with respect to the version controlled package file. If: <ul style="list-style-type: none"> • False, the package data that exists in the model is accepted as being up-to-date and no attempt is made to re-import data from the package file • True, Enterprise Architect re-imports the package from the package file regardless. <p>See also the version control menu option Get Latest in <i>Version Control Within UML Models Using Enterprise Architect</i>.</p>
VersionControlGetStatus ()	<i>Long</i>	<p>Returns the version control status of the package. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p> <p>Return value maps to the following enumerated type:</p> <pre>enum EnumCheckOutStatus { csUncontrolled = 0, csCheckedIn, csCheckedOutToThisUser, csReadOnlyVersion,</pre>

Method	Type	Notes
		<p><i>csCheckedOutToAnotherUser</i>, <i>csOfflineCheckedIn</i>, <i>csCheckedOutOfflineByUser</i>, <i>csCheckedOutOfflineByOther</i>, <i>csDeleted</i>, };</p> <p><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.</p> <p><i>csReadOnlyVersion</i> - Package is marked as read-only. An earlier revision of the package has been retrieved from version control.</p> <p><i>csOfflineCheckedOutToThisUser</i> - Indicates that the package was 'checked out' by this user whilst disconnected from version control.</p> <p><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.</p> <p><i>csDeleted</i> - The package file has been deleted from version control.</p>
VersionControlPutLatest (string CheckInComment)	Void	<p>Perform a checkin of the version controlled package, whilst keeping the package checked-out.</p> <p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p> <p>When a package that was previously marked as <i>Checked Out Offline</i>, is successfully 'Put' (checkedin) to version control, that package's flags are updated to clear the <i>Checked Out Offline</i> indicator.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • Comment: String - Log message added to the version controlled file's history (where applicable).
VersionControlRemove ()	Void	<p>Removes version control from the package.</p> <p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p>
VersionControlResynchPackageStatus (boolean ClearSettings)		<p>Synchronizes the version control status of the single object package recorded in your current model with the package status reported by your version control provider. (See <i>Version Control Within UML Models Using Enterprise Architect</i>.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> • ClearSettings: Boolean - used if the package file associated with the specified package is reported by the version control provider as uncontrolled. If ClearSettings is: <ul style="list-style-type: none"> • True, the version control settings are cleared from the package • False, the version control settings remain unchanged.

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	ObjectType 193	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 false 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	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Organization	Package ^[215] : 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 false 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	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Tag	String	Read/Write. Name of the property (Tag Name).

Author Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>PropertyType</i> object after modification or appending a new item. If false 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
Count	<i>Short</i>	Count of items in the list.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	<i>String</i>	The list type (for example, Diagram Types).

Reference Methods

Method	Type	Notes
GetAt (short Index)	<i>String</i>	Get the item at the specified index. Parameters: <ul style="list-style-type: none"> • Index: Short - The index of the item to retrieve from the list.
GetLastError ()	<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 Stereotype tab of the UML Types dialog in Enterprise Architect.
Notes	String	Read/Write. Notes about the stereotype.
Name	String	Read/Write. The stereotype name. Appears in the Stereotype drop list for elements that match the <i>AppliesTo</i> attribute.
ObjectType	ObjectType 193	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 false 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
ActualTime	Long	Read/Write. Time already expended on task, in hours, days or other units.
AssignedTo	String	Read/Write. Person this task is assigned to; that is, the responsible resource.
EndDate	Date	Read/Write. Date task scheduled to finish.
History	String	Read/Write. Memo field to hold, for example, task history or notes.
Name	Variant	Read/Write. Task name.
Notes	Variant	Read/Write. Description of the task.
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Owner	String	Read/Write. The task owner.
Percent	Long	Read/Write. Percent the task is complete.
Phase	String	Read/Write. The phase of the project the task relates to.
Priority	String	Read/Write. Priority associated with this task.
StartDate	Date	Read/Write. Date task is to start.
Status	Variant	Read/Write. Current task status.
TaskID	Long	Read only. Local ID of task.
TotalTime	Long	Read/Write. The total expected time the task might run - in hours, days or some other unit.
Type	String	Read/Write. Sets or returns string representing the type.

Task 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 Task object after modification or appending a new item. If false 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	ObjectType <small>[193]</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

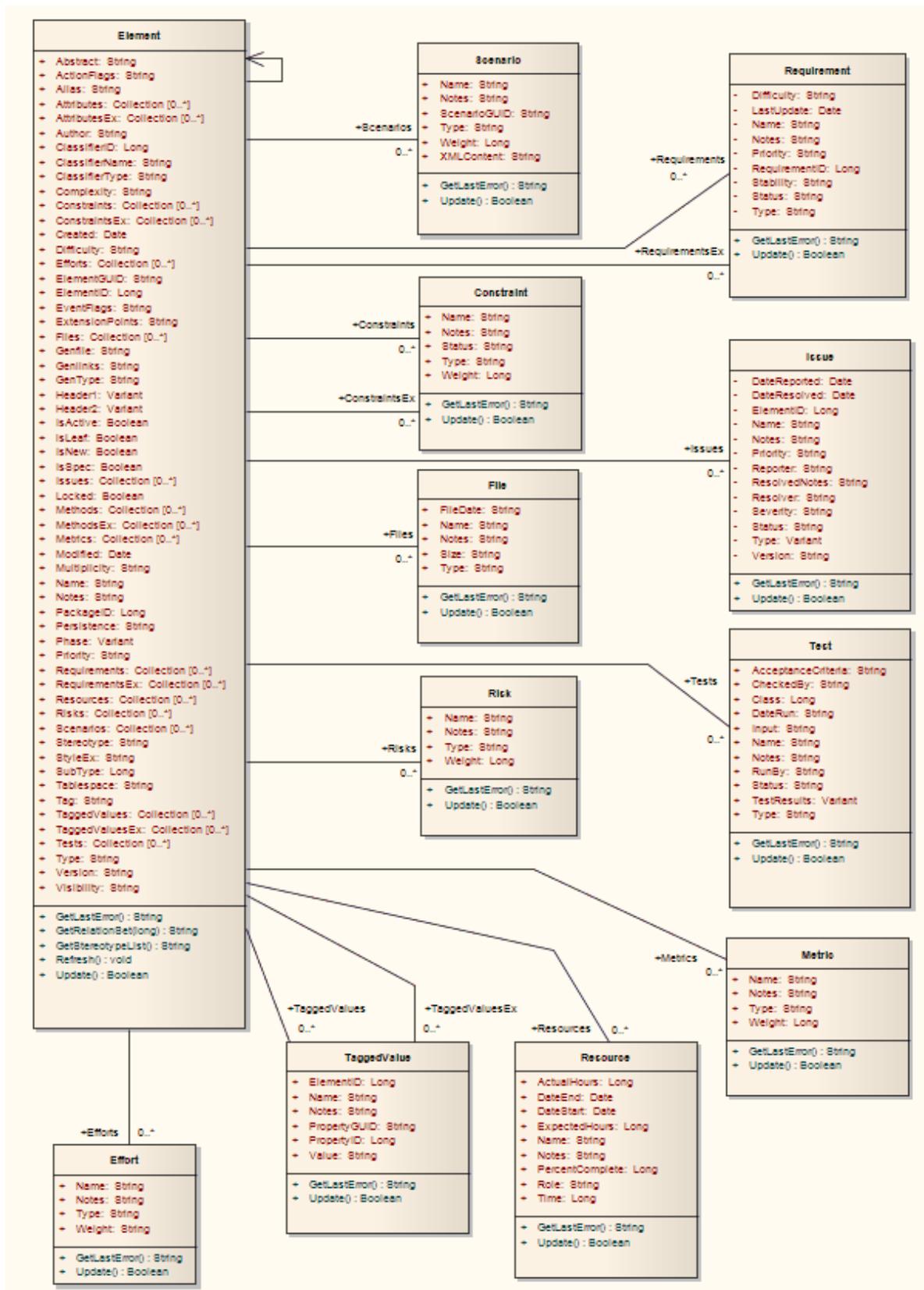
Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Term object after modification or appending a new item. If false 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	<i>String</i>	Read/Write. The name of the constraint (that is, the constraint).
Notes	<i>String</i>	Read/Write. Notes about the constraint.
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
ParentID	<i>Long</i>	Read only. The <i>ElementID</i> of the element to which this constraint applies.
Status	<i>String</i>	Read/Write. Current status.
Type	<i>String</i>	Read/Write. Constraint type.
Weight	<i>Long</i>	Read/Write. A weighting factor.

Constraint Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>Constraint</i> object after modification or appending a new item. If false 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	<i>String</i>	Read/Write. The name of the effort.
Notes	<i>String</i>	Read/Write. Notes about the effort.
ObjectType	ObjectType [193]	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	Collection ^[21]	Read only. Collection of Attribute objects for current element. Use the AddNew and Delete functions to manage attributes.
AttributesEx	Collection ^[21]	Read only. Collection of Attribute objects belonging to the current element and its parent elements.
Author	String	Read/Write. The element author (see the Repository: Authors ^[19] list for more details).
BaseClasses	Collection ^[21]	Read only. List of Base Classes for this element presented as a collection for convenience.
ClassifierID	Long	Deprecated. 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).
ClassifierName	<i>String</i>	Read/Write. Name of associated Classifier (if any).
ClassifierType	<i>String</i>	Read only. Type of associated classifier.
Complexity	<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: 1 for Easy, 2 for Medium, 3 for Difficult.
CompositeDiagram	Diagram ^[264]	Read only. If the element is Composite, returns its associated diagram; otherwise returns null.
Connectors	Collection ^[211]	Read only. Returns a collection containing the connectors to other elements.
Constraints	Collection ^[211]	Read only. Collection of Constraint ^[227] objects.
ConstraintsEx	Collection ^[211]	Read only. Collection of Constraint objects belonging to the current element and its parent elements.
Created	<i>Date</i>	Read/Write. The date the element was created.
CustomProperties	Collection ^[211]	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.
Diagrams	Collection ^[211]	Read only. Returns a collection of sub-diagrams (child diagrams) attached to this element as seen in the tree view.
Difficulty	<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: Low , Medium , High .
Efforts	Collection ^[211]	Read only. Collection of Effort ^[227] objects.
ElementGUID	<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.
ElementID	<i>Long</i>	Read only. The local ID of the Element. Valid for this file only.
Elements	Collection ^[211]	Read only. Returns a collection of child elements (sub-elements) attached to this element as seen in the tree view.
EmbeddedElements	Collection ^[211]	Read only. List of elements that are embedded into this element, such as Ports, Parts, Pins and Parameter Sets.
EventFlags	<i>String</i>	Read/Write. A structure to hold a variety of flags to do with signals or events.
ExtensionPoints	<i>String</i>	Read/Write. Optional extension points for a Use Case as a comma-separated list.
Files	Collection ^[211]	Read only. Collection of File ^[235] objects.
GenFile	<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.
Genlinks	<i>String</i>	Read/Write. Links to other Classes discovered at code reversing time; Parents and Implements connectors only.
GenType	<i>String</i>	Read/Write. The code generation type; for example, Java, C++, C#, VBNet, Visual Basic, Delphi.
Header1	<i>Variant</i>	Read/Write. A user defined string for inclusion as header in the source files generated.
Header2	<i>Variant</i>	Read/Write. Same as for Header1 , but used in the CPP source file.
IsActive	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is active or not. 1 = True, 0 = False.
IsLeaf	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is in leaf node or not. 1 = True, 0 = False.
IsNew	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is new or not. 1 = True, 0 = False.
IsSpec	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is a specification or not. 1 = True, 0 = False.
Issues	Collection ^[21]	Read only. Collection of Issue objects.
Locked	<i>Boolean</i>	Read/Write. Indicates if the element has been locked against further change.
MetaType	<i>String</i>	Read only. The element's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
Methods	Collection ^[21]	Read only. Collection of Method objects for current element.
MethodsEx	Collection ^[21]	Read only. Collection of Method objects belonging to the current element and its parent elements.
Metrics	Collection ^[21]	Read only. Collection of Metric elements for current element.
MiscData	<i>String</i>	Read only. This low-level property provides information about the contents of the PData_x fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property. MiscData is zero based, therefore: <ul style="list-style-type: none"> • MiscData(0) corresponds to PData1 • MiscData(1) to PData2 and so on.
Modified	<i>Date</i>	Read/Write. The date the element was last modified.
Multiplicity	<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	ObjectType [193]	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 0 .
Partitions	Collection ^[211]	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 Persistent or Transient .
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: Low , Medium and High .
Properties	Properties ^[254]	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	Collection ^[211]	Read only. List of Interfaces realized by this element for convenience.
Requirements	Collection ^[211]	Read only. Collection of Requirement ^[237] objects.
RequirementsEx	Collection ^[211]	Read only. Collection of Requirement ^[237] objects belonging to the current element and its parent elements.
Resources	Collection ^[211]	Read only. Collection of Resource ^[238] objects for current element.
Risks	Collection ^[211]	Read only. Collection of Risk ^[239] objects.
RunState	String	Read/Write. The object's runstate list as a string.
Scenarios	Collection ^[211]	Read only. Collection of Scenario ^[239] objects for current element.
StateTransitions	Collection ^[211]	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 Proposed or Approved .
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.								
StereotypeEx	String	Read/Write. All the applied stereotypes of the element in a comma-separated list.								
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.								
Subtype	Long	<p>Read/Write. A numeric subtype that qualifies the Type^[232] of the main element. For example:</p> <ul style="list-style-type: none"> For Event: 0 = Receiver, 1 = Sender For Class: 1 = Parameterised, 2 = Instantiated, 3 = Both, 0 = Neither, 17 = Association Class <p>Note:</p> <p>If 17, because an Association Class has been created through the user interface, MiscData(3) 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"> For Note: 1 = Note linked to connector, 2 = Constraint linked to connector For StateNode: 100 = ActivityInitial, 101 = ActivityFinal For Activity: 0 = Activity, 8 = composite Activity (also set to 8 for other composite elements such as Use Cases) For Synchronization: 0 = Horizontal, 1 = Vertical. <p>Note that there are many more Types than indicated in the above examples.</p>								
Tablespace	String	Read/Write. Associated tablespace for a Table element.								
Tag	String	Read/Write. Corresponds to the Keywords 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> .								
TaggedValues	Collection ^[211] of type TaggedValue ^[242]	Read only. Returns a collection of TaggedValue ^[242] objects.								
TaggedValuesEx	Collection ^[211] of type TaggedValue ^[242]	Read only. Returns a collection of TaggedValue ^[242] objects belonging to the current element and the elements specialized or realized by the current element.								
Tests	Collection ^[211]	Read only. Collection of Test ^[243] objects for current element.								
TreePos	Long	Read/Write. Sets or gets the tree position.								
Type	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" data-bbox="730 1883 1433 2031"> <tbody> <tr> <td>Action</td> <td>InteractionOccurrence</td> </tr> <tr> <td>Activity</td> <td>InteractionState</td> </tr> <tr> <td>ActivityPartition</td> <td>Interface</td> </tr> <tr> <td>ActivityRegion</td> <td>InterruptibleActivityRegion</td> </tr> </tbody> </table>	Action	InteractionOccurrence	Activity	InteractionState	ActivityPartition	Interface	ActivityRegion	InterruptibleActivityRegion
Action	InteractionOccurrence									
Activity	InteractionState									
ActivityPartition	Interface									
ActivityRegion	InterruptibleActivityRegion									

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>
Version	<i>String</i>	Read/Write. The version of the element.
Visibility	<i>String</i>	Read/Write. The Scope of this element within the current package. Valid values are: Public , Private , Protected or Package .

Element Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	<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"> aGroupName: String - the name of the user group for which to set the group lock.
ApplyUserLock ()	<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.
GetLastError ()	<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.
GetLinkedDocument ()	<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.
GetRelationSet	<i>String</i>	Returns a string containing a comma-separated list of

Method	Type	Notes
(EnumRelationSetType Type)		<p>ElementIDs of directly- <i>and indirectly</i>-related elements based on the given type. See EnumRelationSetType^[192].</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 <i>only</i> the direct relationships of the element, use the Connector^[255] collection instead.</p>
GetStereotypeList ()	<i>String</i>	Returns a comma-separated list of stereotypes allied to this element.
LoadLinkedDocument (string Filename)	<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"> • FileName: String - the name of the file from which to load the RTF document.
Refresh ()	<i>Void</i>	Refreshes the element features in the Project Browser . Usually called after adding or deleting attributes or methods, when the user interface is required to be updated as well.
ReleaseUserLock ()	<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>
SaveLinkedDocument (string Filename)	<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"> • FileName: String - the name of the RTF file to which to save the linked document.
SetAppearance (long Scope, long Item, long Value)	<i>Void</i>	<p>Sets the visual appearance of the element.</p> <p>Parameter:</p> <ul style="list-style-type: none"> • Scope: Long - Scope of appearance set to modify <ul style="list-style-type: none"> 0 – Local (Diagram-local appearance) 1 – Base (Default appearance across entire model) • Item: Long - Appearance item to modify <ul style="list-style-type: none"> 0 – Background color 1 – Font Color 2 – Border Color 3 – Border Width • Value: Long - Value to set appearance to.
Update ()	<i>Boolean</i>	<p>Update the current element object after modification or appending a new item.</p> <p>If false 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	<i>String</i>	Read/Write. The file date when entry is created.
Name	<i>String</i>	Read/Write. The file name can be a logical file or a reference to a web address (using <i>http://</i>).
Notes	<i>String</i>	Read/Write. Notes about the file.
ObjectType	ObjectTyp <small>e^[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Size	<i>String</i>	Read/Write. The file size.
Type	<i>String</i>	Read/Write. File type.

File Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current File object after modification or appending a new item. If false 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	<i>Date</i>	Read/Write. Date issue reported.
DateResolved	<i>Date</i>	Read/Write. Date issue resolved.
ElementID	<i>Long</i>	Read/Write. ID of element associated with this issue.
Name	<i>String</i>	Read/Write. The Issue name; that is, the Issue itself.
Notes	<i>String</i>	Read/Write. Issue description.

Attribute	Type	Notes
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Priority	String	Read/Write. Issue priority. Generally should use Low , Medium and High .
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 Low , Medium or High .
Status	String	Read/Write. The current status of the issue.
Type	Variant	Read/Write. Issue type - can be Defect or Change , Issue and ToDo .
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 false 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	ObjectType [193]	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
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current Metric object after modification or appending a new item. If false 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
Difficulty	<i>String</i>	Read/Write. Estimated difficulty to implement.
LastUpdate	<i>Date</i>	Read/Write. Date requirement last updated.
Name	<i>String</i>	Read/Write. The requirement itself.
Notes	<i>String</i>	Read/Write. Further notes about requirement.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
ParentID	<i>Long</i>	Read only. The <i>ElementID</i> of the element to which this requirement applies.
Priority	<i>String</i>	Read/Write. Assigned priority of the requirement.
RequirementID	<i>Long</i>	Read only. A local ID for this requirement.
Stability	<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.
Status	<i>String</i>	Read/Write. Current status of the requirement.
Type	<i>String</i>	Read/Write. Requirement type.

Requirement Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current Requirement object after modification or appending a new item. If false 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
ActualHours	<i>Long</i>	Read/Write. Time already expended on the task, in hours, days or other units.
DateEnd	<i>Date</i>	Read/Write. Expected end date.
DateStart	<i>Date</i>	Read/Write. Date to start work.
ExpectedHours	<i>Long</i>	Read/Write. The total expected time the task might run, in hours, days or other units.
History	<i>String</i>	Read/Write. Gets or sets history text.
Name	<i>String</i>	Read/Write. Name of resource (for example, person's name).
Notes	<i>String</i>	Read/Write. Descriptive notes.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
PercentComplete	<i>Long</i>	Read/Write. Current percent complete figure.
Role	<i>String</i>	Read/Write. Role they play in implementing the element.
Time	<i>Long</i>	Read/Write. Time expected; numeric indicating number of days.

Resource Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current Resource object after modification or appending a new item. If false 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
Name	<i>String</i>	Read/Write. The risk.
Notes	<i>String</i>	Read/Write. Further notes describing the risk.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	<i>String</i>	Read/Write. The risk type associated with this element.
Weight	<i>Long</i>	Read/Write. A weighting for estimation or metric purposes.

Risk Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current Risk object after modification or appending a new item. If false 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
Name	<i>String</i>	Read/Write. The Scenario name.
Notes	<i>String</i>	Read/Write. Description of the Scenario. Usually contains the steps to execute the scenario.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
ScenarioGUID	<i>String</i>	Read/Write. A unique ID for the Scenario. Used to identify the Scenario unambiguously within a model.
Steps	<i>Collection of ScenarioStep</i> <small>[217]</small>	Read only. A collection of step objects for this Scenario. Use the AddNew and <i>Delete</i> functions to manage steps. <i>AddNew</i> passes the step

Attribute	Type	Notes
	241	name and "1" as the type for an actor step.
Type	String	Read/Write. The scenario type (for example, <i>Basic Path</i>).
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. It is recommended that you use the Steps ^[239] collection to read or modify this field.

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 false is returned, check the <i>GetLastError</i> function for more information.

7.2.5.11 ScenarioExtension

ScenarioExtension Attributes

Attribute	Type	Notes
ExtensionGUID	String	Read/Write. A unique GUID for this Extension.
Join	String	Read/Write. The GUID of the step where this Extension rejoins the Scenario.
JoiningStep	ScenarioStep ^[241]	Read only. The actual step where this Extension rejoins the Scenario, if any.
Level	String	Read only. The number of this Extension as shown in the scenario editor. This is derived from the value of <i>Pos</i> for this object and the owning step.
Name	String	Read/Write. The Extension name. Note: This should match the name of the linked scenario.
ObjectType	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Pos	Long	Read/Write. The position of the Extension in the Extensions list
Scenario	Scenario ^[239]	Read only. The scenario that is executed as an alternative path for this Extension.

ScenarioExtension Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>ScenarioExtension</i> object after modification or appending a new item. If false is returned, check the <i>GetLastError</i> function for more information.

7.2.5.12 ScenarioStep

ScenarioStep Attributes

Attribute	Type	Notes
Extensions	<i>Collection of ScenarioExtension</i> ^[240]	Read only. A collection of <i>ScenarioExtension</i> objects that specify how the scenario is extended from this step. The arguments to AddNew ^[211] should match the name and GUID of the alternative scenario being linked to.
Level	<i>String</i>	Read only. The number of this Step as shown in the scenario editor. This is derived from the value of <i>Pos</i> .
Link	<i>String</i>	Read/Write. The GUID of a Use Case that is relevant to this step.
LinkedElement	<i>Element</i> ^[228]	Read only. The actual element specified by Link, if any.
Name	<i>String</i>	Read/Write. The Step name.
ObjectType	<i>ObjectType</i> ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Pos	<i>Long</i>	Read/Write. The position of the Step in the Scenario Step list.
Results	<i>String</i>	Read/Write. Any results that are given from this step.
State	<i>String</i>	Read/Write. A description of the state the system enters when this Step is executed.
StepGUID	<i>String</i>	Read/Write. A unique GUID for this Step.
StepType	<i>ScenarioStepType</i> ^[195]	Read/Write. Identifies whether this step is being performed by a user or the system.
Uses	<i>String</i>	Read/Write. Input and requirements that are relevant to this step.

ScenarioStep Methods

Method	Type	Notes
GetLastError ()	<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
Update ()	Boolean	Update the current <i>ScenarioStep</i> object after modification or appending a new item. If false is returned, check the <i>GetLastError</i> function for more information.

7.2.5.13 TaggedValue

public Class

A *TaggedValue* is a named property and value associated with an element. It 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 tag.
Notes	String	Read/Write. Further descriptive notes about this tag. If Value (below) is set to " <memo> ", then Notes should contain the actual Tagged Value content.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String	Read/Write. The tag global ID.
PropertyID	Long	Read only. The tag local ID.
Value	String	Read/Write. The value assigned to this tag. This field has a 255 character limit. If the value is greater than 255 characters long, set the value to " <memo> " and insert the body of text in the Notes attribute (above). When reading existing Tagged Values, if Value = " <memo> " then the developer should read the actual body of text from the Notes attribute.

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 <i>TaggedValue</i> object after modification or appending a new item. If false is returned, check the <i>GetLastError</i> function for more information.

7.2.5.14 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	ObjectType [193]	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 false is returned, check the <i>GetLastError</i> function for more information.

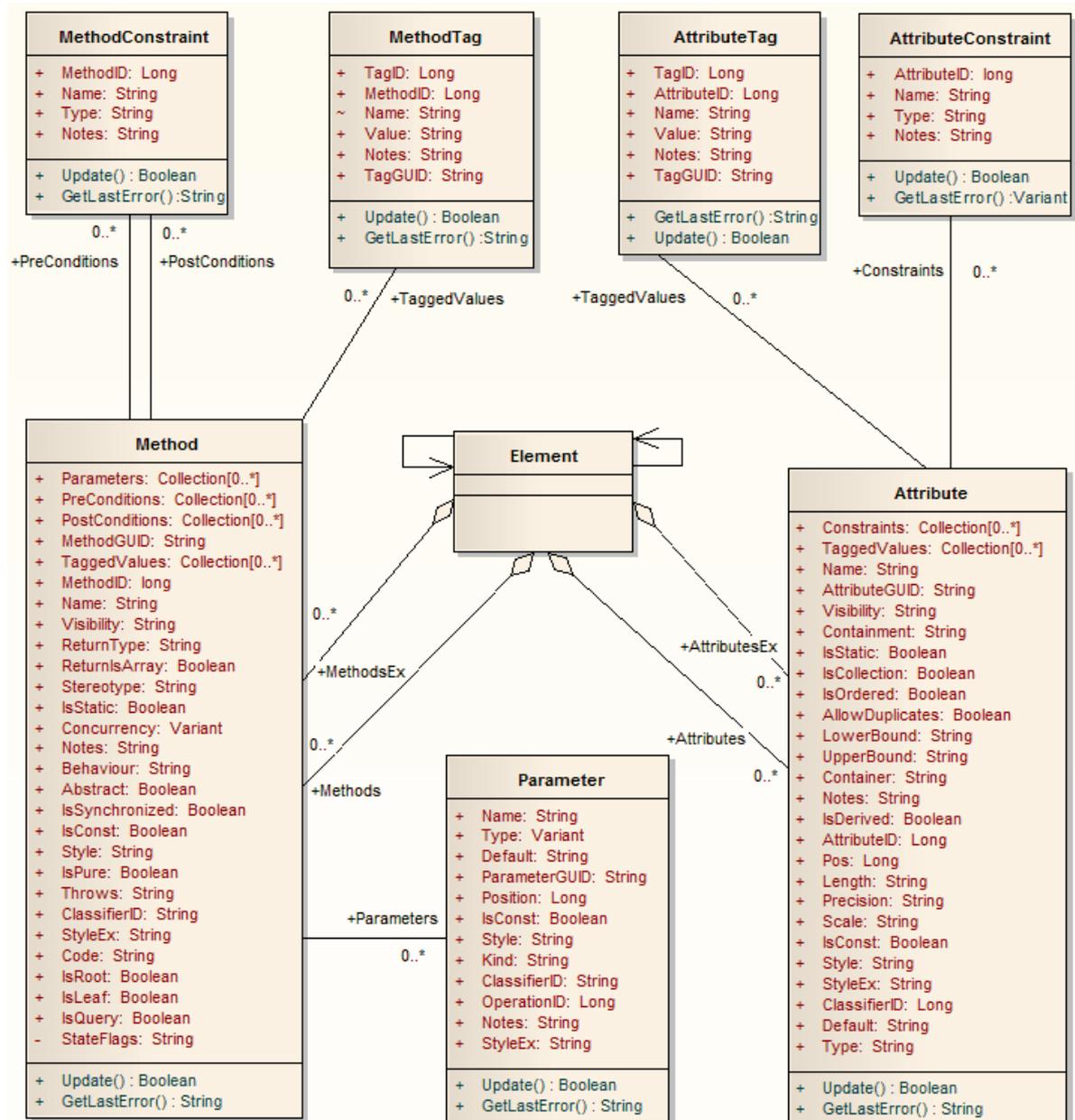
7.2.6 Element Features

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
AllowDuplicat s	<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 Not Null option.
AttributeGUID	<i>String</i>	Read/Write. A globally unique ID for the current attribute. System generated.
AttributeID	<i>Long</i>	Read only. Local ID number of the attribute.
ClassifierID	<i>Long</i>	Read/Write. Classifier ID, if appropriate; indicates the base type associated with attribute, if not a primitive type.
Container	<i>String</i>	Read/Write. The container type.
Containment	<i>String</i>	Read/Write. Type of containment. Can be Not Specified, By Reference or By Value .
Constraints	Collection <small>[21†]</small>	Read only. A collection of <i>AttributeConstraint</i> objects. Used to access and manage constraints associated with this attribute.
Default	<i>String</i>	Read/Write. Initial value assigned to this attribute.
IsCollection	<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.
IsConst	<i>Boolean</i>	Read/Write. Flag indicating if the attribute is Const or not.
IsDerived	<i>Boolean</i>	Read/Write. Indicates if the attribute is derived (that is, a calculated value).
IsOrdered	<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.
IsStatic	<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 Unique option.
Length	<i>String</i>	Read/Write. The attribute length, where applicable.
LowerBound	<i>String</i>	Read/Write. A value for the collection lower bound.
Name	<i>String</i>	Read/Write. The attribute name.
Notes	<i>String</i>	Read/Write. Further notes about this attribute.
ObjectType	ObjectType <small>[193†]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
ParentID	<i>Long</i>	Read only. Returns the <i>ElementID</i> of the element that this attribute is a part of.
Pos	<i>Long</i>	Read/Write. Position of the attribute in the Class attribute list.
Precision	<i>String</i>	Read/Write. Precision value.
Scale	<i>String</i>	Read/Write. Scale value.
Stereotype	<i>String</i>	Read/Write. Sets or gets the stereotype for this attribute.
StereotypeEx	<i>String</i>	Read/Write. All the applied stereotypes of the attribute in a comma-separated list.

Attribute	Type	Notes
Style	String	Read/Write. Contains the Alias property for this attribute.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
TaggedValues	Collection [21†] of type AttributeTag [24†]	Read only. A collection of AttributeTag objects. Use to access and manage Tagged Values associated with this attribute.
TaggedValuesEx	Collection [21†] of type TaggedValue e[242†]	Read only. Collection of TaggedValue objects belonging to the current attribute and the TaggedValuesEx property of its classifier.
Type	String	Read/Write. The attribute type (by name; also see <i>ClassifierID</i>).
UpperBound	String	Read/Write. A value for the collection upper bound.
Visibility	String	Read/Write. The scope of the attribute. Can be Private , Protected , Public or Package .

Attribute 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 attribute object after modifying or appending a new item. If false 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
AttributeID	Long	Read/Write. ID of the attribute this constraint applies to.
Name	String	Read/Write. The constraint.
Notes	String	Read/Write. Descriptive notes about constraint.
ObjectType	ObjectType [193†]	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	String	Read/Write. Type of constraint.

AttributeConstraint Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>AttributeConstraint</i> object after modification or appending a new item. If false 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
AttributeID	<i>Long</i>	Read/Write. Local ID of attribute associated with this Tagged Value.
Name	<i>String</i>	Read/Write. Name of tag.
Notes	<i>String</i>	Read/Write. Further descriptive notes about this tag. If Value (below) is set to " <memo> ", then Notes should contain the actual Tagged Value content.
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
TagGUID	<i>String</i>	Read/Write. A globally unique ID for this Tagged Value.
TagID	<i>Long</i>	Read only. Local ID to identify Tagged Value.
Value	<i>String</i>	Read/Write. The value assigned to this tag. This field has a 255 character limit. If the value is greater than 255 characters long, set the value to " <memo> " and insert the body of text in the Notes attribute (above). When reading existing Tagged Values, if Value = " <memo> " then the developer should read the actual body of text from the Notes attribute.

AttributeTag Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>AttributeTag</i> object after modification or appending a new item. If false is returned, check the <i>GetLastError</i> function for

Method	Type	Notes
		more information.

7.2.6.4 CustomProperties

public Collection

The *CustomProperties* collection contains 0 or more *Cust 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

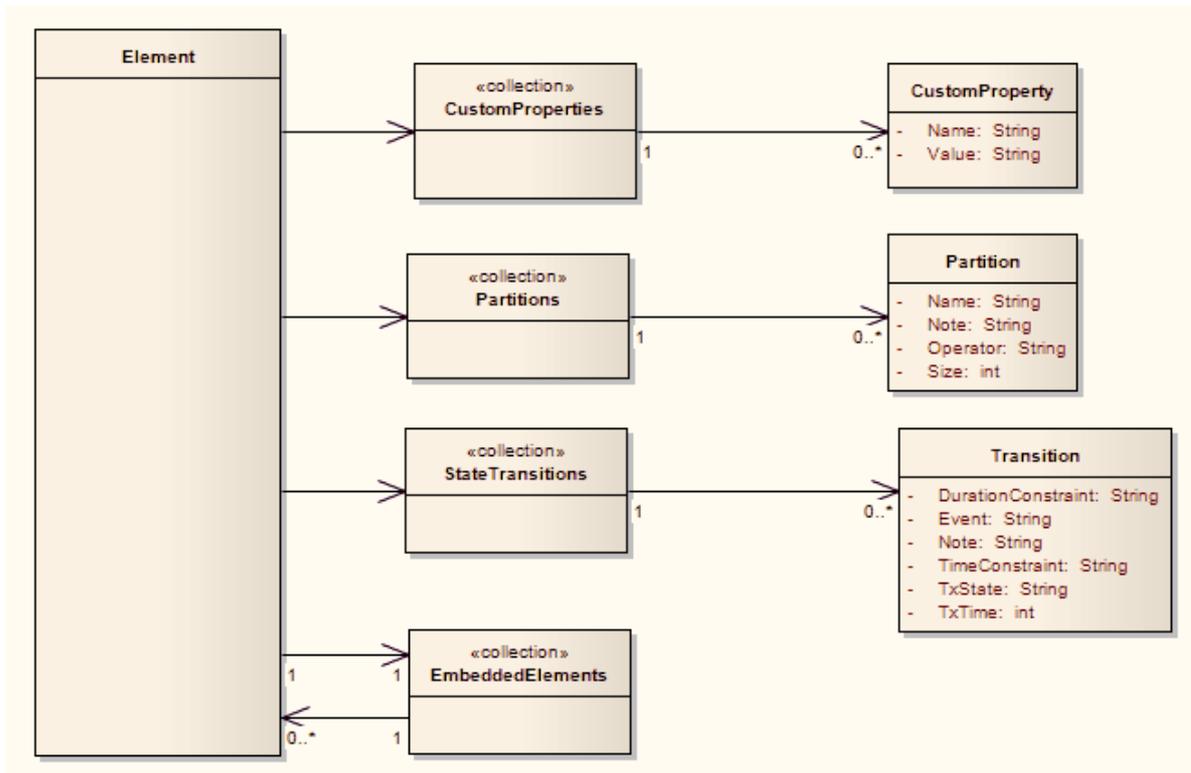
Attribute	Type	Notes
Name	<i>String</i>	Read only. The CustomProperty name.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Value	<i>String</i>	Read/Write. The value associated with this custom property. Can be a string, the boolean values true or false , 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
Abstract	<i>Boolean</i>	Read/Write. Flag indicating if the method is abstract (1) or not (0).
Behavior	<i>String</i>	Read/Write. Some further explanatory behavior notes (for example, pseudocode). Note: 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.
ClassifierID	<i>String</i>	Read/Write. Classifier ID that applies to the <i>ReturnType</i> .
Code	<i>String</i>	Read/Write. Optional field to hold the method Code (used for the Initial Code field).
Concurrency	<i>Variant</i>	Read/Write. Concurrency type of method.
IsConst	<i>Boolean</i>	Read/Write. Flag indicating the method is Const .

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	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Parameters	Collection ^[21]	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. Returns the <i>ElementID</i> of the element that this method belongs to.
Pos	Long	Read/Write. Specifies the position of the method within the set of operations defined for a Class.
PostConditions	Collection ^[21]	Read only. PostConditions (constraints) as they apply to this method. Returns a <i>MethodConstraint</i> object of type post .
PreConditions	Collection ^[21]	Read only. PreConditions (constraints) as they apply to this method. Returns a <i>MethodConstraint</i> object of type pre .
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 Alias property for this method.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
TaggedValues	Collection ^[21] of type MethodTag <small>[252]</small>	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: Public , Protected , Private or Package .

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 false 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	ObjectType [193]	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 false 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. Further descriptive notes about this tag. If Value (below) is set to "<memo>", then Notes should contain the actual Tagged Value content.
ObjectType	ObjectType [193]	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. The value assigned to this tag. This field has a 255 character limit. If the value is greater than 255 characters long, set the value to "<memo>" and insert the body of text in the Notes attribute (above). When reading existing Tagged Values, if Value = "<memo>" then the developer should read the actual body of text from the Notes attribute.

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 false 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).
Kind	String	Read/Write. The parameter kind - in , inout , out , return .
Name	String	Read/Write. The parameter name; must be unique for a single method.
Notes	String	Read/Write. Descriptive notes.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
OperationID	Long	Read only. ID of the method associated with this parameter.
ParameterGUID	String	Read/Write. A globally unique ID for the current Parameter. System generated.
Position	Long	Read/Write. The position in the argument list.
Stereotype	String	Read/Write. The first stereotype of the parameter.
StereotypeEx	String	Read/Write. All the applied stereotypes of the parameter in a comma-separated list.
Style	String	Read/Write. Some style information.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Type	Variant	Read/Write. The parameter type; can be a primitive type or defined classifier.

Parameter 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 Parameter object after modifying or appending a new item. If false 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	String	Read/Write. The partition name; can represent a condition or constraint in some cases.
Note	String	Read/Write. A free text note associated with this partition.
ObjectType	ObjectType ^[193]	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	ObjectType ^[193]	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"> Index: Variant - either a string representing the property name or an integer representing the zero-based offset into the property list.

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 Value is identical to that obtained through the automation property.
ObjectType	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	PropType ^[194]	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

Attribute	Type	Notes
		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.

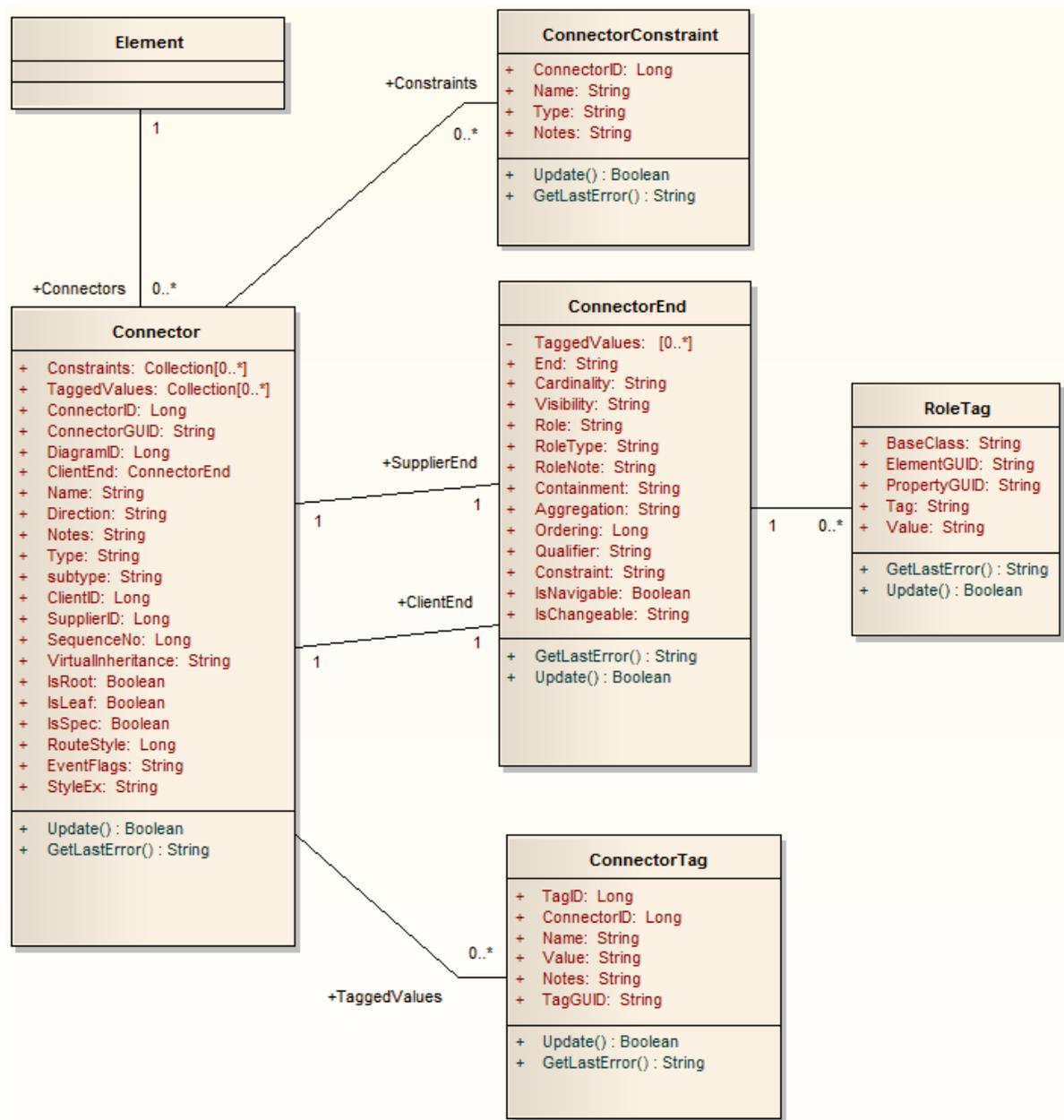
Transition Attributes

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	ObjectType [193]	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 Timeline Properties 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	ObjectType <small>[193]</small>	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 false 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
- Assembly
- Association
- Collaboration
- CommunicationPath
- Connector
- ControlFlow
- Delegate
- Dependency
- Deployment
- ERLink
- Generalization
- InformationFlow
- Instantiation
- InterruptFlow
- Manifest
- Nesting
- NoteLink
- ObjectFlow
- Package
- Realization
- Sequence
- StateFlow
- UseCase

Associated table in .EAP file: *t_connector*

Connector Attributes

Attribute	Type	Notes
Alias	String	Read/Write. An optional alias for this connector.
ClientEnd	ConnectorEnd ^[260]	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.
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	Collection ^[217]	Read only. Collection of constraint ^[227] objects.
CustomProperties	Collection ^[217]	Read only. Returns a collection of advanced properties associated with an element in the form of CustomProperty ^[248] 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"> • Unspecified • Bi-Directional • Source -> Destination • Destination -> Source
EndPointX	Long	Read/Write. The x-coordinate of the connector's end point. Note: 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. Note: 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.
MiscData	String	Read only. This low-level property provides information about the contents of the PDatax fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property.

Attribute	Type	Notes
		<p>MiscData is zero based, therefore:</p> <ul style="list-style-type: none"> • MiscData(0) corresponds to PData1 • MiscData(1) to PData2 <p>and so on.</p>
Name	<i>String</i>	Read/Write. The connector name.
Notes	<i>String</i>	Read/Write. Descriptive notes about the connector.
ObjectType	ObjectType [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Properties	Properties [254]	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	<i>Long</i>	Read/Write. The route style.
SequenceNo	<i>Long</i>	Read/Write. The <i>SequenceNo</i> of the connector.
StartPointX	<i>Long</i>	<p>Read/Write. The x-coordinate of the connector's start point.</p> <p>Note:</p> <p>Connector end points are specified in Cartesian coordinates with the origin to the top left of the screen.</p>
StartPointY	<i>Long</i>	<p>Read/Write. The y-coordinate of the connector's start point.</p> <p>Note:</p> <p>Connector end points are specified in Cartesian coordinates with the origin to the top left of the screen.</p>
StateFlags	<i>String</i>	Read/Write. Structure to hold a variety of flags concerned with State signaling on messages, the list delimited by semi-colons.
Stereotype	<i>String</i>	Read/Write. Sets or gets the stereotype for this connector end.
StereotypeEx	<i>String</i>	Read/Write. All the applied stereotypes of the connector in a comma-separated list.
StyleEx	<i>String</i>	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Subtype	<i>String</i>	Read/Write. A possible subtype to refine the meaning of the connector.
SupplierEnd	ConnectorEnd [260]	Read only. A pointer to the <i>ConnectorEnd</i> object representing the target end of the relationship.
SupplierID	<i>Long</i>	Read/Write. <i>ElementID</i> of the element at the target end of this connector.
TaggedValues	<i>Collection</i>	Read only. Collection of <i>ConnectorTag</i> objects.
TransitionAction	<i>String</i>	Read/Write. See the <i>Transition</i> topic in the <i>UML Dictionary</i> for appropriate values.
TransitionEvent	<i>String</i>	Read/Write. See the <i>Transition</i> topic in the <i>UML Dictionary</i> for

Attribute	Type	Notes
		appropriate values.
TransitionGuard	String	Read/Write. See the <i>Transition</i> topic in the <i>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.
MiscData (long Index)	String	Read only. This low-level property provides information about the contents of the PData fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property. MiscData is zero based, therefore: <ul style="list-style-type: none"> • MiscData(0) corresponds to PData1 • MiscData(1) to PData2 Parameters: <ul style="list-style-type: none"> • Index: long - the zero based index of the PData field to access.
Update ()	Boolean	Update the current <i>ConnectorObject</i> after modification or appending a new item. If false 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
Aggregation	Long	Read/Write. Aggregation as it applies to this end. Valid values are: 0 = None 1 = Shared 2 = Composite.
Alias	String	Read/Write. An optional alias for this connector end.
AllowDuplicates	Boolean	Read/Write. For multiplicities greater than 1, indicates that duplicate

Attribute	Type	Notes
		entries are possible.
Cardinality	<i>String</i>	Read/Write. Cardinality associated with this end.
Constraint	<i>String</i>	Read/Write. A constraint that can be applied to this connector end.
Containment	<i>String</i>	Read/Write. Containment type applied to this connector end.
Derived	<i>Boolean</i>	Read/Write. Indicates that the value of this end is derived.
DerivedUnion	<i>Boolean</i>	Read/Write. Indicates the value of this role derived from the union of all roles that subset this.
End	<i>String</i>	Read only. The end this <i>ConnectorEnd</i> object applies to: <i>Client</i> or <i>Supplier</i> .
IsChangeable	<i>String</i>	Read/Write. Flag indicating whether this end is changeable or not. Values: frozen , addOnly or none .
IsNavigable	<i>Boolean</i>	Read/Write. Flag indicating this end is navigable from the other end.
Navigable	<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-Navigable</i> and <i>Unspecified</i> .
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Ordering	<i>Long</i>	Read/Write. Ordering for this connector end.
OwnedByClassifier	<i>Boolean</i>	Read/Write. Indicates this Association end corresponds to an attribute on the opposite end of the Association.
Qualifier	<i>String</i>	Read/Write. A qualifier that can apply to connector end.
Role	<i>String</i>	Read/Write. The connector end role.
RoleNote	<i>String</i>	Read/Write. Notes associated with the role of this connector end.
RoleType	<i>String</i>	Read/Write. The role type applied to this end of the connector.
Stereotype	<i>String</i>	Read/Write. Sets or gets the stereotype for this connector end.
StereotypeEx	<i>String</i>	Read/Write. All the applied stereotypes of the connector end in a comma-separated list.
TaggedValues	<i>Private</i>	Read only. Collection of RoleTag <small>[263]</small> objects.
Visibility	<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
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>ConnectorEnd</i> object after modification or

Method	Type	Notes
		appending a new item. If false 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
ConnectorID	Long	Read/Write. The local ID of the associated connector.
Name	String	Read/Write. The tag or name.
Notes	String	Read/Write. Further descriptive notes about this tag. If Value (below) is set to " <memo> ", then Notes should contain the actual Tagged Value content.
ObjectType	ObjectType [193]	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. The value assigned to this tag. This field has a 255 character limit. If the value is greater than 255 characters long, set the value to " <memo> " and insert the body of text in the Notes attribute (above). When reading existing Tagged Values, if Value = " <memo> " then the developer should read the actual body of text from the Notes attribute.

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 false 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 ASSOCIATION_SOURCE or ASSOCIATION_TARGET .
ElementGUID	String	Read/Write. GUID of the connector with which this role tag is associated.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String	Read/Write. A system generated GUID to identify the Tagged Value.
Tag	String	Read/Write. The actual tag name.
Value	String	Read/Write. The value associated with this tag.

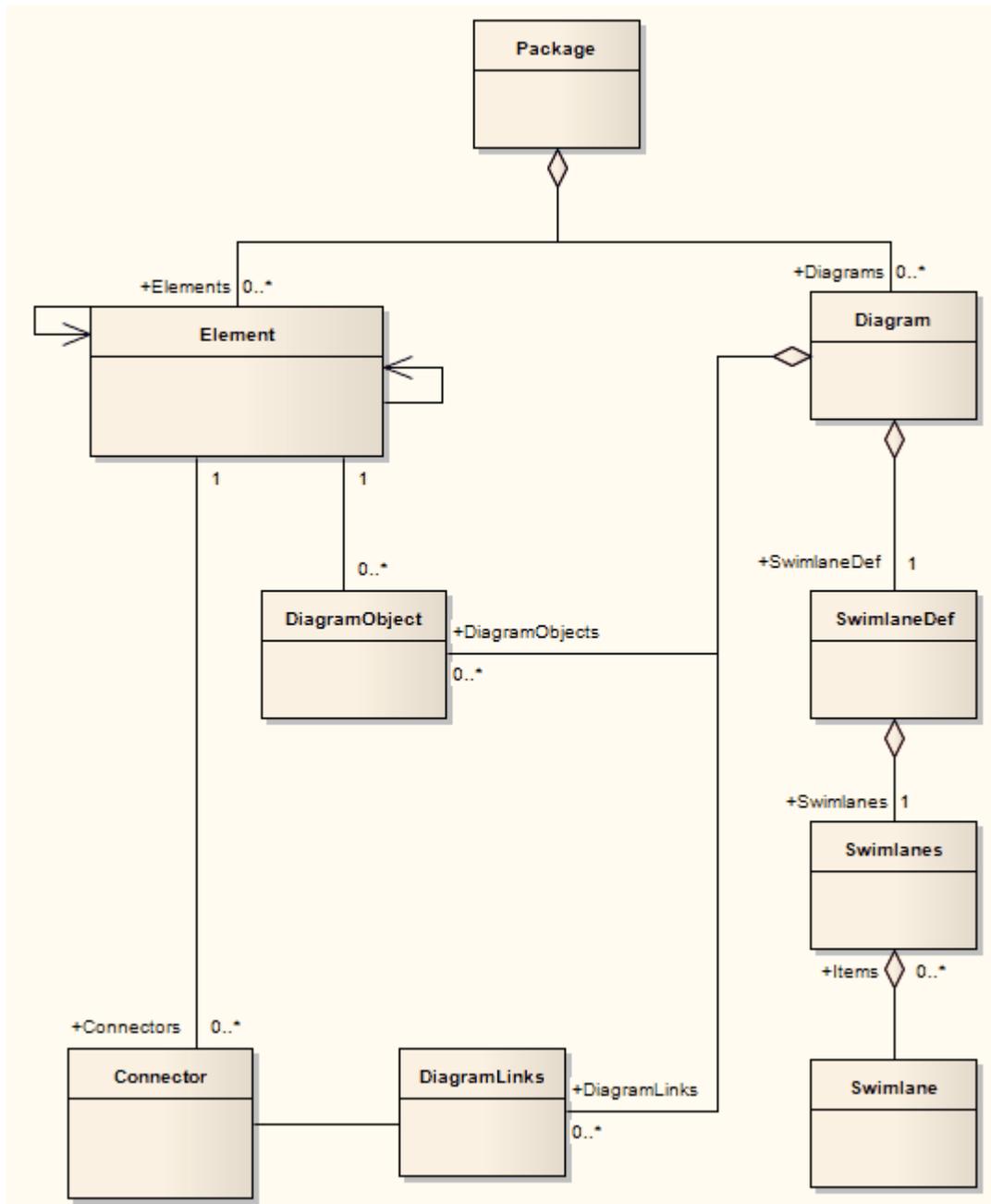
RoleTag Methods

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 RoleTag after changes or on initial creation. If false 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	Collection [21†]	Read only. A list of <i>DiagramLink</i> objects, each containing information about the display characteristics of a connector in a diagram. Note: A DiagramLink 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 DiagramLink is not in use.
DiagramObjects	Collection [21†]	Read only. A collection of references to DiagramObjects [268]. 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	ObjectType e [193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Orientation	String	Read/Write. Page orientation: P for Portrait or L for Landscape.
PackageID	Long	Read/Write. An ID of the package that this diagram belongs to.
ParentID	Long	Read/Write. An optional ID of an element that 'owns' this diagram; for

Attribute	Type	Notes
		example, a Sequence diagram owned by a Use Case.
Scale	Long	Read/Write. The zoom scale (default is 100).
SelectedConnector	Connector <small>[257]</small>	Read/Write. The currently selected connector on this diagram. Null if there is no currently selected diagram.
SelectedObjects	Collection <small>[217]</small>	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. 1 = Show, 0 = 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 SwimlaneDef <small>[269]</small> .
SwimlaneDef	SwimlaneDef <small>[269]</small>	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"> aGroupName: String - the name of the user group for which to set the group lock.
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 occurs.

Method	Type	Notes
ReleaseUserLock ()	<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.
ReorderMessages ()	<i>Void</i>	Resets the display order of Sequence and Collaboration messages. Typically used after inserting or deleting messages in the diagram.
ShowAsElementList (bool ShowAsList, bool Persist)	<i>Boolean</i>	Toggles the diagram display between diagram format and Element List 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"> • ShowAsList: Boolean - indicates diagram or Element List • Persist: Boolean - indicates set (maintain <i>ShowAsList</i> value) or not (revert to default).
Update ()	<i>Boolean</i>	Updates this diagram object after modification or appending a new item. If false 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](#)^[265] collection.

Associated table in .EAP file: *t_diagramlinks*

DiagramLinks Attributes

Attribute	Type	Notes
ConnectorID	<i>Long</i>	Read/Write. The ID of the associated connector.
DiagramID	<i>Long</i>	Read/Write. The local ID for the associated diagram.
Geometry	<i>String</i>	Read/Write. The geometry associated with the current connector in this diagram.
InstanceID	<i>Long</i>	Read only. Holds the connector identifier for the current model.
IsHidden	<i>Boolean</i>	Read/Write. Flag to indicate if this item is hidden or not.
ObjectType	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.
Path	<i>String</i>	Read/Write. The path of the connector in this diagram.
Style	<i>String</i>	Read/Write. Additional style information; for example, color, thickness.

DiagramLinks Methods

Method	Type	Notes
GetLastError ()	<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.
Update ()	<i>Boolean</i>	Update the current <i>DiagramLink</i> object after modification or appending a new item. If false 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
Bottom	<i>Long</i>	Read/Write. The bottom position of the element.
DiagramID	<i>Long</i>	Read/Write. The ID of the associated diagram (long).
ElementID	<i>Long</i>	Read/Write. The <i>ElementID</i> of the object instance in this diagram.
InstanceID	<i>Long</i>	Read/Write. Read only attribute. Holds the connector identifier for the current model.
Left	<i>Long</i>	Read/Write. The left position of the element.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Right	<i>Long</i>	Read/Write. The right position of the element.
Sequence	<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.
Style	<i>Variant</i>	Write only (reading this value gives undefined results). Style information for this object. See Setting the Style <small>[269]</small> below for more information.
Top	<i>Long</i>	Read/Write. The top position of the element.

DiagramObjects Methods

Method	Type	Notes
GetLastError ()	<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
Update ()	<i>Boolean</i>	Update the current <i>DiagramObject</i> object after modification or appending a new item. If false 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
Bold	<i>Boolean</i>	Read/Write. Show the title text in bold.
FontColor	<i>Long</i>	Read/Write. RGB color used to draw the titles.
HideClassifier	<i>Boolean</i>	Read/Write. Removes any classifier from title display.
HideNames	<i>Boolean</i>	Read/Write. Set to true to hide the swimlane titles.
LineColor	<i>Long</i>	Read/Write. RGB color used to draw swimlane borders.
LineWidth	<i>Long</i>	Read/Write. Width of line, in pixels, used to draw swimlanes. Valid values: 1 , 2 or 3 .
Locked	<i>Boolean</i>	Read/Write. If set to true, disables user modification of the swimlanes via the diagram.
ObjectType	ObjectType 193	Read only. Distinguishes objects referenced through a Dispatch interface.
Orientation	<i>String</i>	Read/Write. Indication of whether the swimlanes are vertical or horizontal.
ShowInTitleBar	<i>Boolean</i>	Read/Write. Enables vertical swimlane titles to be shown in title bar.

Attribute	Type	Notes
Swimlanes	Swimlanes <small>[270]</small>	Read/Write. A list of individual swimlanes.

7.2.8.5 Swimlanes

A *Swimlanes* object is attached to a diagram's [SwimlaneDef](#)^[269] object and provides a mechanism to access individual swimlanes.

Swimlanes Attributes

Attribute	Type	Notes
Count	Long	Read/Write. Gives the number of swimlanes.
ObjectType ^[193]	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.

Swimlanes Methods

Method	Type	Notes
Add (string Title, long Width)	Swimlane ^[271]	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"> Title: String - The title text that appears at the top of the swimlane. Can be the same as an existing swimlane title. Width: Long - The width of the swimlane in pixels.
Delete (object Index)	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"> Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to delete.
DeleteAll ()	Void	Removes all swimlanes.
Insert (long Index, string Title, long Width)	Swimlane ^[271]	Inserts a swimlane at a specific position. Returns a swimlane object representing the newly added entry. Parameters: <ul style="list-style-type: none"> Index: Long - The zero-based index of the existing Swimlane before which this new entry is inserted. Title: String - The title text which appears at the top of the swimlane. Can be the same as an existing swimlane title. Width: Long - The width of the swimlane in pixels.
Items (object Index)	Swimlane ^[271] <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"> Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to get.

7.2.8.6 Swimlane

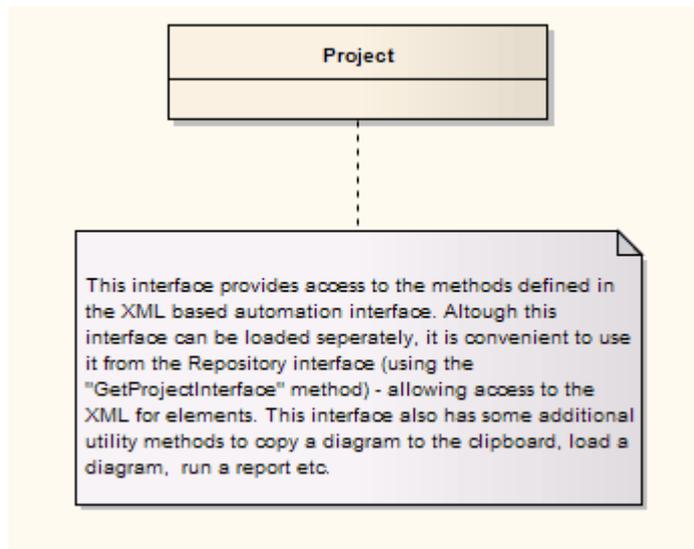
A *Swimlane* object makes available attributes relating to a single row or column in a list of [swimlanes](#) ^[270].

Attribute	Type	Notes
BackColor	<i>Long</i>	Read/Write. The swimlane is filled with this RGB color.
ClassifiedGuid	<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.
ObjectType	ObjectType <small>[193]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Title	<i>String</i>	Read/Write. Text at the head of the swimlane.
Width	<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](#) ^[278] to change the Enterprise Architect GUID to an XML GUID.

Project Attributes

Attribute	Type	Notes
ObjectType	ObjectType ^[193]	Read only. Distinguishes objects referenced through a Dispatch interface.

Project Methods

Method	Type	Notes
CreateBaseline (string PackageGUID, string Version, string Notes)	<i>Boolean</i>	Creates a Baseline of a specified package. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to Baseline. Version: String - the version of the Baseline. Notes: String - any notes concerning the Baseline.
CreateBaselineEx (string PackageGUID, string Version, string Notes, EA.CreateBaselineFlag Flags)	<i>Boolean</i>	Creates a Baseline of a specified package, with a flag to exclude package contents below the first level. (See <i>Baseline UML Models</i> .) Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to Baseline. Version: String - the version of the Baseline. Notes: String - any notes concerning the Baseline. Flags: EA.CreateBaselineFlag^[192] - whether or not to exclude package contents below the first level
DefineRule (string CategoryID, EA.EnumMVErrorType ErrorType, string ErrorMessage)	<i>String</i>	Defines the individual rules that can be performed during model validation. It must be called once for each rule from the EA_OnInitializeUserRules ^[160] 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 Model Validation Example ^[165] for a detailed example of use of this method. Parameters: <ul style="list-style-type: none"> CategoryId: String - should be passed the return value from the DefineRuleCategory^[272] method. ErrorType: EA.EnumMVErrorType - depending on the severity of the error being validated, can be: <ul style="list-style-type: none"> mvErrorCritical mvError mvWarning, or mvInformation. ErrorMessage: String - can contain a default error string, although this is probably overridden by the PublishResult^[281] call.
DefineRuleCategory (string	<i>String</i>	Defines a category of rules that can be performed during model validation (there is typically one category

Method	Type	Notes
CategoryName)		<p>per Add-In). It must be called once from the EA_OnInitializeUserRules^[160] broadcast handler.</p> <p>The return value is a CategoryId that must to be passed to the DefineRule^[272] method.</p> <p>See Model Validation Example^[165] for a detailed example of use of this method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • CategoryName: String - a text string that is visible in the Model Validation Configuration dialog.
DeleteBaseline (string BaselineGUID)	Boolean	<p>Deletes a Baseline, identified by the BaselineGUID, from the repository.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • BaselineGUID: String - the GUID (in XML format) of the Baseline to delete.
DoBaselineCompare (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 GetBaselines^[277]).</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 DoBaselineMerge^[273] - automatically merging information from the Baseline.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • PackageGUID: String - the GUID (in XML format) of the package to run the comparison on. • Baseline: String - the GUID (in XML format) of the Baseline to run the comparison on. • ConnectString: String - the location of the external .EAP file or DBMS to extract the Baseline from.
DoBaselineMerge (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><Merge> <MergeItem guid="{XXXXXX}" /> <MergeItem guid="{XXXXXX}" /> </Merge></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><Merge></pre>

Method	Type	Notes
		<pre><MergeItem guid="RestoreAll" /></pre> <pre></Merge></pre> Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to merge the Baseline into. Baseline: String - the GUID of the Baseline (in XML format) to merge into the package. MergeInstructions: String - the file containing the GUID of each differenced item from the XML difference log returned by DoBaselineCompare()^[273]. ConnectionString: String - the location of the EAP file or DBMS to get the Baseline from, if not in the same model as the package.
EnumDiagramElements (string DiagramGUID)	protected abstract: <i>String</i>	Gets an XML list of all elements in a diagram. Parameters: <ul style="list-style-type: none"> DiagramGUID: String - the GUID (in XML format) of the diagram to get elements for.
EnumDiagrams (string PackageGUID)	protected abstract: <i>String</i>	Gets an XML list of all diagrams in a specified package. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to list diagrams for.
EnumElements (string PackageGUID)	protected abstract: <i>String</i>	Gets an XML list of elements in a specified package. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to get a list of elements for.
EnumLinks (string ElementGUID)	protected abstract: <i>String</i>	Gets an XML list of connectors for a specified element. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element to get all associated connectors for.
EnumPackages (string PackageGUID)	protected abstract: <i>String</i>	Gets an XML list of child packages inside a parent package. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the parent package.
EnumProjects ()	protected abstract: <i>String</i>	Gets a list of projects in the current file; corresponds to Models ^[198] in <i>Repository</i> .
EnumViews ()	protected abstract: <i>String</i>	Enumerates the Views for a project. Returned as an XML document.
EnumViewEx (string ProjectGUID)	protected abstract: <i>String</i>	Gets a list of Views in the current project. Parameters: <ul style="list-style-type: none"> ProjectGUID: String - the GUID (in XML format) of the project to get views for.
Exit ()	protected abstract: <i>String</i>	Exits the current instance of Enterprise Architect; this function is maintained for backward compatibility and

Method	Type	Notes
		<p>should never be called.</p> <p>Enterprise Architect automatically exits when you are no longer using any of the provided objects.</p>
ExportPackageXML (string PackageGUID, enumXMIMType, long DiagramXML, long DiagramImage, long FormatXML, long UseDTD, string FileName)	protected abstract: <i>String</i>	<p>Exports XML for a specified package.</p> <p>Parameters:</p> <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to be exported. XMIMType: EnumXMIMType - specifies the XML type and version information; see XMIMType Enum^[195] for accepted values. DiagramXML: Long - true if XML for diagrams is required; accepted values: 0 = Do not export diagrams 1 = Export diagrams 2 = Export diagrams along with alternate images. 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. FormatXML: Long - true if XML output should be formatted prior to saving. UseDTD: Long - true if a DTD should be used. FileName: String - the filename to output to.
ExportPackageXMIMEx (string PackageGUID, enumXMIMType, long DiagramXML, long DiagramImage, long FormatXML, long UseDTD, string FileName, ea. ExportPackageXMIMFlag Flags)	protected abstract: <i>String</i>	<p>Exports XML for a specified package, with a flag to determine whether the export includes package content below the first level. (See the <i>Controlled Packages</i> section in <i>UML Model Management</i>.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to be exported. XMIMType: EnumXMIMType - specifies the XML type and version information; see XMIMType Enum^[195] for accepted values. DiagramXML: Long - true if XML for diagrams is required; accepted values: 0 = Do not export diagrams 1 = Export diagrams 2 = Export diagrams along with alternate images. 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. FormatXML: Long - true if XML output should be formatted prior to saving.

Method	Type	Notes
		<ul style="list-style-type: none"> UseDTD: Long - true if a DTD should be used. FileName: String - the filename to output to. Flags: ea.ExportPackageXMLFlag^[193] - whether or not to include package content below the first level (currently only supported for <i>xmiEADefault</i>).
GenerateClass (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"> ElementGUID: String - the GUID (in XML format) of the element to generate. ExtraOptions: String - enables extra options to be given to the command; currently unused.
GenerateDiagramFromScenario (string ElementGUID, EnumScenarioDiagramType DiagramType, long OverwriteExistingDiagram)	<i>Boolean</i>	<p>Generates various diagrams from the Structured Specification of an element.</p> <p>Parameters:</p> <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element containing the Structured Specification. DiagramType: EnumScenarioDiagramType - the type of diagram to generate; see ScenarioDiagramType Enum^[194] for accepted values OverwriteExistingDiagram: Long - determines whether to overwrite the existing diagram or synchronize the existing elements with the scenario steps <ul style="list-style-type: none"> 0 = Delete existing diagram and elements, and create new diagram and elements 1 = Synchronize existing elements with scenario steps and preserve diagram layout 2 = Synchronize existing elements with scenario steps and re-cast diagram layout 3 = Do not generate diagram if one already exists.
GeneratePackage (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"> PackageGUID: String - the GUID (in XML format) of the package to generate. ExtraOptions: String - enables extra options to be given to the command; currently enables: <ul style="list-style-type: none"> Generation of all subpackages (<i>recurse</i>) Force overwrite of all files (<i>overwrite</i>) and Specification to auto generate all paths (<i>dir</i>).
GenerateTestFromScenario (string ElementGUID, EnumScenarioTestType TestType)	<i>Boolean</i>	<p>Generates either an Internal test or an External test from the Structured Specification of an element. (See the <i>Scenarios</i> section of <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i>.)</p> <p>Parameters:</p> <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element containing the Structured Specification. TestType: EnumScenarioTestType - the type of test to generate; see ScenarioTestType Enum^[195]

Method	Type	Notes
		for accepted values.
GenerateXSD (string PackageGUID, string FileName, string Encoding, string Options)	<i>Boolean</i>	Creates an XML schema for this GenerateXSD. Returns true on success. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package. FileName: String - target filepath. Encoding: String - the XML encoding for the code page instruction. Options: String - enables extra options to be given to the command; currently enables: <ul style="list-style-type: none"> <i>GenGlobalElement</i> - turn the generation of global elements for all global <i>ComplexTypes</i> On or Off; for example: - GenGlobalElement=1.
GetBaselines (string PackageGUID, string ConnectString)	<i>String</i>	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). Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to get Baselines for. ConnectString: String - the location of the EAP file or DBMS to get the Baselines from, if not in the same model as the package.
GetDiagram (string DiagramGUID)	protected abstract: <i>String</i>	Gets diagram details, in XML format. Parameters: <ul style="list-style-type: none"> DiagramGUID: String - the GUID (in XML format) of the diagram to get details for.
GetElement (string ElementGUID)	protected abstract: <i>String</i>	Gets XML for the specified element. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element to retrieve XML for.
GetElementConstraints (string ElementGUID)	protected abstract: <i>String</i>	Gets constraints for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element.
GetElementEffort (string ElementGUID)	protected abstract: <i>String</i>	Gets efforts for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element.
GetElementFiles (string ElementGUID)	protected abstract: <i>String</i>	Gets metrics for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element.
GetElementMetrics (string ElementGUID)	protected abstract: <i>String</i>	Gets files for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format)

Method	Type	Notes
		of the element.
GetElementProblems (string ElementGUID)	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"> ElementGUID: String - the GUID (in XML format) of the element.
GetElementProperties (string ElementGUID)	protected abstract: <i>String</i>	Gets Tagged values for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element.
GetElementRequirements (string ElementGUID)	protected abstract: <i>String</i>	Gets a list of requirements for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String -the GUID (in XML format) of the element.
GetElementResources (string ElementGUID)	protected abstract: <i>String</i>	Gets a list of resources for an element, in XML format. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element.
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"> ElementGUID: String - the GUID (in XML format) of the element.
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"> ElementGUID: String - the GUID (in XML format) of the element.
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"> ElementGUID: String - the GUID (in XML format) of the element.
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"> LinkGUID: String - the GUID (in XML format) of the connector to get details of.
GUIDtoXML (string GUID)	<i>String</i>	Changes an internal GUID to the form used in XML. Parameters: <ul style="list-style-type: none"> GUID: String - the Enterprise Architect style GUID to convert to XML format.
ImportDirectory (string)	<i>Boolean</i>	Imports a source code directory into the model.

Method	Type	Notes
PackageGUID, string Language, string DirectoryPath, string ExtraOptions)		Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to reverse engineer code into. Language: String - specifies the language of the code to be imported. DirectoryPath: String - specifies the path where the code is found on the computer. 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>.
ImportFile (string PackageGUID, string Language, string FileName, string ExtraOptions)	Boolean	Imports an individual file or binary module into the model, in a package per namespace style import. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to reverse engineer code into; this is expected to be a namespace root package. Language: String - specifies the language of the code to be imported. <p>Note: Use the value "DNPE" to import a binary module. This imports a .Net assembly or Java .class file, but not a .jar file.</p> <ul style="list-style-type: none"> Filename: String - specifies the path where the code or module is found on the computer. ExtraOptions: String - enables extra options to be given to the command; currently unused.
ImportPackageXML (string PackageGUID, string Filename, long ImportDiagrams, long StripGUID)	String	Imports an XMI file at a point in the tree. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the target package to import the XMI file into (or overwrite with the XMI file). Filename or XMLText: String - the name of the XMI file. <p>Note: 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"> ImportDiagrams: Long. 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.
LayoutDiagram (string DiagramGUID, long LayoutStyle)	Boolean	Deprecated. it is recommended that <i>LayoutDiagramEx</i> is used instead. Calls the function to automatically layout a diagram in hierarchical fashion. It is only recommended for Class and Object diagrams. Parameters:

Method	Type	Notes
		<ul style="list-style-type: none"> DiagramGUID: String - the GUID (in XML format) of the diagram to lay out. LayoutStyle: Long - always ignored.
LayoutDiagramEx (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 ConstLayoutStyles Enum ^[19†]):</p> <ul style="list-style-type: none"> Default Options: <ul style="list-style-type: none"> IsDiagramDefault IsProgramDefault. Cycle Removal Options: <ul style="list-style-type: none"> IsCycleRemoveGreedy IsCycleRemoveDFS. Layering Options: <ul style="list-style-type: none"> IsLayeringLongestPathSink IsLayeringLongestPathSource IsLayeringOptimalLinkLength. Initialize Options: <ul style="list-style-type: none"> IsInitializeNaive IsInitializeDFSOut IsInitializeDFSIn. Crossing Reduction Option: <ul style="list-style-type: none"> IsCrossReduceAggressive. Layout Options - Direction <ul style="list-style-type: none"> IsLayoutDirectionUp IsLayoutDirectionDown IsLayoutDirectionLeft IsLayoutDirectionRight. <p>Parameters:</p> <ul style="list-style-type: none"> DiagramGUID: String - the GUID (in XML format) of the diagram to lay out. LayoutStyle: Long - the layout style. Iterations: Long - the number of layout iterations the Layout process should take to perform cross reduction (Default value = 4). LayerSpacing: Long - the per-element layer spacing the Layout process shall use (Default value = 20). ColumnSpacing: Long - the per-element column spacing the Layout process shall use (Default value = 20). SaveToDiagram: Boolean - specifies whether or not Enterprise Architect should save the supplied layout options as default to the diagram in question.
LoadControlledPackage (string PackageGUID)	<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"> PackageGUID: String - the GUID (in XML format) of the package to load.

Method	Type	Notes
LoadDiagram (string DiagramGUID)	protected abstract: <i>Boolean</i>	Loads a diagram by its GUID. Parameter: <ul style="list-style-type: none"> DiagramGUID: String - the GUID (in XML format) of the diagram to load; if you retrieve the GUID using the Diagram interface, use the GUIDtoXML [278] function to convert it to XML format.
LoadProject (string FileName)	protected abstract: <i>Boolean</i>	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. Parameters: <ul style="list-style-type: none"> FileName: String - the name of the project file to load.
MigrateToBPMN11 (string GUID, string Type)	<i>Void</i>	Migrates every BPMN 1.0 construct in a package or an element (including elements, attributes, diagrams and connectors) to BPMN 1.1. Parameters <ul style="list-style-type: none"> GUID: String - the GUID of the package or element for which the contents are to be migrated to BPMN 1.1 Type: String - the type of upgrade, either just to BPMN 1.1 or to BPMN 1.1 and BPEL. Accepted values: <p>BPMN = migrate to BPMN 1.1 BPEL = migrate to BPMN 1.1 and update:</p> <ul style="list-style-type: none"> any diagram with stereotype <i>BPMN</i> to <i>BPEL</i> any element with stereotype <i>BusinessProcess</i> to <i>BPELProcess</i>. <p>Note:</p> <p>Migrating to BPEL is possible only in the Ultimate or Business and Software Engineering editions of Enterprise Architect.</p>
PublishResult (string CategoryID, EA.EnumMVErrortype ErrorType, string ErrorMessage)	<i>String</i>	Returns the results of each rule that can be performed during model validation. It must be called once for each rule from the EA_OnInitializeUserRules [165] 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 Model Validation Example [165] for a detailed example of use of this method. Parameters: <ul style="list-style-type: none"> CategoryId: String - should be passed the return value from the DefineRuleCategory [272] method. ErrorType: EA.EnumMVErrortype - depending on the severity of the error being validated, can be: <ul style="list-style-type: none"> mvErrorCritical mvError mvWarning, or

Method	Type	Notes
		<ul style="list-style-type: none"> • mvInformation. • ErrorMessage: String - contains an error string.
PutDiagramImageOnClipboard (string DiagramGUID, long Type)	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"> • DiagramGUID: String - the GUID (in XML format) of the diagram to copy. • Type: Long - the file type. <ul style="list-style-type: none"> • If Type = 0 then it is a metafile • If Type = 1 then it is a Device Independent Bitmap.
PutDiagramImageToFile (string Diagram GUID, string FileName, long Type)	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"> • DiagramGUID: String - the GUID (in XML format) of the diagram to save. • FileName: String - the name of the file to save the diagram into. • Type: Long - the file type. <ul style="list-style-type: none"> • If type = 0 then it is a metafile • If type = 1 then it uses the file type from the name extension (that is, .bmp, .jpg, .gif, .png, .tga)
ReloadProject ()	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>
RunReport (string PackageGUID, string TemplateName, string Filename)	protected abstract: <i>Void</i>	<p>Runs a named RTF report.</p> <p>Parameters:</p> <ul style="list-style-type: none"> • PackageGUID: String - the GUID (in XML format) of the package to run the report on. • TemplateName: String - the RTF report template to use. If the PackageGUID has a stereotype of <i>MasterDocument</i>, the template is not required. • FileName: String - the file name to store the generated report in.
RunHTMLReport (string PackageGUID, string ExportPath, string ImageFormat, string Style, string Extension)	<i>String</i>	<p>Runs an HTML report (same as Documentation HTML Documentation when you right-click on a package in the Project Browser).</p> <p>Parameters:</p> <ul style="list-style-type: none"> • PackageGUID: String - the GUID (in XML format) of the package to run the report on. • ExportPath: String - the file name to store the generated report in. • ImageFormat: String. • Style: String. • Extension: String.
SaveControlledPackage (string PackageGUID)	<i>String</i>	<p>Saves a package that has been configured as a controlled package, to XMI. Only the package GUID is required, Enterprise Architect picks the rest up from the package control information.</p>

Method	Type	Notes
		Parameter: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package to save.
SaveDiagramImageToFile (string Filename)	protected abstract: <i>String</i>	Saves a diagram image of the current diagram to file. Parameters: <ul style="list-style-type: none"> FileName: String - the filename of the image to save.
ShowWindow (long Show)	protected abstract: <i>Void</i>	Shows or hides the Enterprise Architect User Interface. Parameters: <ul style="list-style-type: none"> Show: Long.
SynchronizeClass (string ElementGUID, string ExtraOptions)	<i>Boolean</i>	Synchronizes a Class with the latest source code. Parameters: <ul style="list-style-type: none"> ElementGUID: String - the GUID (in XML format) of the element to update from code. ExtraOptions: String - enables extra options to be given to the command; currently unused.
SynchronizePackage (string PackageGUID, string ExtraOptions)	<i>Boolean</i>	Synchronizes each Class in a package with the latest source code. Parameters: <ul style="list-style-type: none"> PackageGUID: String - the GUID (in XML format) of the package containing the elements to update from code. 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>.
TransformElement (string TransformName, string ElementGUID, string TargetPackage, string ExtraOptions)	<i>Boolean</i>	Transforms an element into a package. Parameters: <ul style="list-style-type: none"> TransformName: String - specifies the transformation that should be executed. ElementGUID: String - the GUID (in XML format) of the element to transform. TargetPackageGUID: String - the GUID (in XML format) of the package to transform into. ExtraOptions: String - enables extra options to be given to the command; currently unused.
TransformPackage (string TransformName, string SourcePackage, string TargetPackage, string ExtraOptions)	<i>Boolean</i>	Runs a transformation on the contents of a package. Parameters: <ul style="list-style-type: none"> TransformName: String - specifies the transformation that should be executed. SourcePackageGUID: String - the GUID (in XML format) of the package to transform. TargetPackageGUID: String - the GUID (in XML format) of the package to transform into. ExtraOptions: String - enables extra options to be given to the command; currently unused.
XMLtoGUID (string GUID)	<i>String</i>	Changes a GUID in XML format to the form used inside Enterprise Architect.

Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> • GUID: String - the XML style GUID to convert to Enterprise Architect internal format.

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](#) ^[284]
- [Iterate Through a .EAP File](#) ^[284]
- [Add and Manage Packages](#) ^[285]
- [Add and Manage Elements](#) ^[286]
- [Add a Connector](#) ^[286]
- [Add and Manage Diagrams](#) ^[287]
- [Add and Delete Features](#) ^[288]
- [Element Extras](#) ^[288]
- [Repository Extras](#) ^[290]
- [Stereotypes](#) ^[292]
- [Work with Attributes](#) ^[292]
- [Work with Methods](#) ^[293]

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

    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
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
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
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
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
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
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
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
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

- . -

.NET

Garbage Collect (Exit Method) 197

- A -

Add

Code Modules In MDG Technology Wizard 40

Diagram Type In MDG Technology Wizard 36

Enumeration Tags To Stereotypes 14

Images In MDG Technology Wizard 43

Linked Document Template In MDG Technology Wizard 45

MDA Transforms In MDG Technology Wizard 42

Pattern In MDG Technology Wizard 35

Profile In MDG Technology Wizard 35

RTF Report Template In MDG Technology Wizard 44

Scripts In MDG Technology Wizard 43

Shape Script To Stereotype In Profile 15

Tagged Value Types In MDG Technology Wizard 39

Task Panel In MDG Technology Wizard 38

Toolbox In MDG Technology Wizard 37

Add And Delete Attributes

Automation Interface Code Example 288

Add And Delete Methods

Automation Interface Code Example 288

Add And Manage Diagrams

Automation Interface Code Example 287

Add And Manage Elements

Automation Interface Code Example 286

Add And Manage Packages

Automation Interface Code Example 285

Add Connector

Automation Interface Code Example 286

Add Stereotypes

Automation Interface Code Example 292

Add-In

And Enterprise ArchitectDeadlocks (.NET) 126

COM Interoperability 126

Concurrent Method Calls 126

Create 124

Create, Define Menu Items 124

Deploy 125

Disable 128

Enable 128

Events 129

Holding State Information 126

Manage 128

Manager 128

Pre-2004 126

Re-entrancy 126

Run Functions From Tasks Pane 56

Search 128

Search Data 129

Tasks 124

Visual Basic Issues 126

Add-In Event

EA_Connect 130

EA_Disconnect 130

EA_GetMenuItems 131

EA_GetmenuState 131

EA_MenuClick 132

EA_OnOutputItemClicked 133

EA_OnOutputItemDoubleClicked 133

EA_ShowHelp 134

Add-In Model

Add-In Event, EA_Connect 130

Add-In Event, EA_Disconnect 130

Add-In Event, EA_GetMenuItems 131

Add-In Event, EA_GetMenuState 131

Add-In Event, EA_MenuClick 132

Add-In Event, EA_OnOutputItemClicked 133

Add-In Event, EA_OnOutputItemDoubleClicked 133

Add-In Event, EA_ShowHelp 134

Add-In Event, Overview 129

Add-In Tasks 124

Benefits 123

Broadcast Event, EA_FileClose 136

Broadcast Event, EA_FileNew 136

Broadcast Event, EA_FileOpen 135

Broadcast Event, EA_OnPostCloseDiagram 137

Broadcast Event, EA_OnPostInitialized 151

Broadcast Event, EA_OnPostOpenDiagram 137

Broadcast Event, EA_OnPostTransform 151

Broadcast Event, EA_OnPreExitInstance 146

Broadcast Event, EA_OnRetrieveModelTemplate 168

Broadcast Events 135

Compartment Events 158

Compartment Events, EA_GetCompartmentData 159

Compartment Events, EA_QueryAvailableCompartments 158

Context Item Events 155

- Add-In Model
 - Context Item Events, EA_OnContextItemChanged 155, 157
 - Context Item Events, EA_OnContextItemDoubleClicked 156
 - Create Add-In 124
 - Create Add-In, Tricks and Traps 126
 - Create Custom View 169
 - Custom View 169
 - EA_Connect 130
 - EA_Disconnect 130
 - EA_FileClose 136
 - EA_FileNew 136
 - EA_FileOpen 135
 - EA_GetCompartmentData 159
 - EA_GetMenuItems 131
 - EA_GetMenuState 131
 - EA_MenuClick 132
 - EA_OnContextItemChanged 155, 157
 - EA_OnContextItemDoubleClicked 156
 - EA_OnDeleteTechnology 154
 - EA_OnEndValidation 161
 - EA_OnImportTechnology 155
 - EA_OnInitializeTechnologies 152
 - EA_OnInitializeUserRules 160
 - EA_OnOutputItemClicked 133
 - EA_OnOutputItemDoubleClicked 133
 - EA_OnPostActivateTechnology 153
 - EA_OnPostCloseDiagram 137
 - EA_OnPostInitialized 151
 - EA_OnPostNewAttribute 149
 - EA_OnPostNewConnector 147
 - EA_OnPostNewDiagram 148
 - EA_OnPostNewDiagramObject 148
 - EA_OnPostNewElement 147
 - EA_OnPostNewMethod 149
 - EA_OnPostNewPackage 150
 - EA_OnPostOpenDiagram 137
 - EA_OnPostTransform 151
 - EA_OnPreActivateTechnology 152
 - EA_OnPreDeleteAttribute 138
 - EA_OnPreDeleteConnector 139
 - EA_OnPreDeleteDiagram 140
 - EA_OnPreDeleteElement 138
 - EA_OnPreDeleteMethod 139
 - EA_OnPreDeletePackage 141
 - EA_OnPreDeleteTechnology 153
 - EA_OnPreExitInstance 146
 - EA_OnPreNewAttribute 144
 - EA_OnPreNewConnector 142
 - EA_OnPreNewDiagram 143
 - EA_OnPreNewDiagramObject 143
 - EA_OnPreNewElement 141
 - EA_OnPreNewMethod 145
 - EA_OnPreNewPackage 145
 - EA_OnRetrieveModelTemplate 168
 - EA_OnRunAttributeRule 163
 - EA_OnRunConnectorRule 163
 - EA_OnRunDiagramRule 162
 - EA_OnRunElementRule 162
 - EA_OnRunMethodRule 164
 - EA_OnRunPackageRule 162
 - EA_OnRunParameterRule 164
 - EA_OnStartValidation 161
 - EA_QueryAvailableCompartments 158
 - EA_ShowHelp 134
 - Interface 123
 - Introduction 123
 - MDG Add-Ins 170
 - MDG Add-Ins, MDG Events 170
 - MDG Events, MDG_BuildProject 171
 - MDG Events, MDG_Connect 171
 - MDG Events, MDG_Disconnect 172
 - MDG Events, MDG_GetConnectedPackages 173
 - MDG Events, MDG_GetProperty 173
 - MDG Events, MDG_Merge 174
 - MDG Events, MDG_NewClass 175
 - MDG Events, MDG_PostGenerate 176
 - MDG Events, MDG_PostMerge 176
 - MDG Events, MDG_PreGenerate 177
 - MDG Events, MDG_PreMerge 177
 - MDG Events, MDG_PreReverse 178
 - MDG Events, MDG_RunExe 179
 - MDG Events, MDG_View 179
 - Model Validation Broadcasts 160
 - Model Validation Broadcasts, EA_OnEndValidation 161
 - Model Validation Broadcasts, EA_OnInitializeUserRules 160
 - Model Validation Broadcasts, EA_OnRunAttributeRule 163
 - Model Validation Broadcasts, EA_OnRunConnectorRule 163
 - Model Validation Broadcasts, EA_OnRunDiagramRule 162
 - Model Validation Broadcasts, EA_OnRunElementRule 162
 - Model Validation Broadcasts, EA_OnRunMethodRule 164
 - Model Validation Broadcasts, EA_OnRunPackageRule 162
 - Model Validation Broadcasts, EA_OnRunParameterRule 164
 - Model Validation Broadcasts, EA_OnStartValidation 161

- Add-In Model
 - Model Validation Example 165
 - Post-New Events 146
 - Post-New Events, EA_OnPostNewAttribute 149
 - Post-New Events, EA_OnPostNewConnector 147
 - Post-New Events, EA_OnPostNewDiagram 148
 - Post-New Events, EA_OnPostNewDiagramObject 148
 - Post-New Events, EA_OnPostNewElement 147
 - Post-New Events, EA_OnPostNewMethod 149
 - Post-New Events, EA_OnPostNewPackage 150
 - Pre-Deletion Events 138
 - Pre-Deletion Events, EA_OnPreDeleteAttribute 138
 - Pre-Deletion Events, EA_OnPreDeleteConnector 139
 - Pre-Deletion Events, EA_OnPreDeleteDiagram 140
 - Pre-Deletion Events, EA_OnPreDeleteElement 138
 - Pre-Deletion Events, EA_OnPreDeleteMethod 139
 - Pre-Deletion Events, EA_OnPreDeletePackage 141
 - Pre-New Events 141
 - Pre-New Events, EA_OnPreNewAttribute 144
 - Pre-New Events, EA_OnPreNewConnector 142
 - Pre-New Events, EA_OnPreNewDiagram 143
 - Pre-New Events, EA_OnPreNewDiagramObject 143
 - Pre-New Events, EA_OnPreNewElement 141
 - Pre-New Events, EA_OnPreNewMethod 145
 - Pre-New Events, EA_OnPreNewPackage 145
 - Search Data, XML Format 129
 - Technology Event, EA_OnInitializeTechnologies 152
 - Technology Events 152
 - Technology Events, EA_OnDeleteTechnology 154
 - Technology Events, EA_OnImportTechnology 155
 - Technology Events, EA_OnPostActivateTechnology 153
 - Technology Events, EA_OnPreActivateTechnology 152
 - Technology Events, EA_OnPreDeleteTechnology 153
- App Object
 - Automation Interface 190
- Attribute
 - Add And Delete, Automation Interface Code Example 288
 - Automation Interface, ElementFeatures Package 244
 - Of Toolbox Page 48
 - PData & StyleEx, Diagram Profiles 53
 - Supported, Create Composite Elements 23
 - Supported, Define Child Diagram Types 23
 - Supported, Metatype In UML Profiles 19
 - Supported, Stereotype In UML Profiles 19
 - Work With, Automation Interface Code Example 292
- AttributeConstraint
 - Automation Interface, ElementFeatures Package 246
- AttributeTag
 - Automation Interface, ElementFeatures Package 247
- Author
 - Attributes 210
 - Methods 210
- Author Collection
 - Automation Interface Repository 210
- Automation Interface
 - App Object 190
 - Attribute, ElementFeatures Package 244
 - AttributeConstraint, ElementFeatures Package 246
 - AttributeTag, ElementFeatures Package 247
 - Available Resources 186
 - Call Executables From Enterprise Architect 181
 - Call From Enterprise Architect 185
 - Code Example, Add And Delete Attributes 288
 - Code Example, Add And Delete Methods 288
 - Code Example, Add And Manage Diagrams 287
 - Code Example, Add And Manage Elements 286
 - Code Example, Add And Manage Packages 285
 - Code Example, Add Connector 286
 - Code Example, Add Stereotypes 292
 - Code Example, Iterate Through EAP File 284
 - Code Example, Open The Repository 284
 - Code Example, Use Element Extras 288
 - Code Example, Use Repository Extras 290
 - Code Example, Work With Attributes 292
 - Code Example, Work With Methods 293
 - Code Examples, Introduction 284
 - Connect From Borland Delphi 7.0 181
 - Connect From Java 181
 - Connect From MS C# 181

- Automation Interface
 - Connect From MS Visual Basic 6.0 181
 - Connect To 181
 - Connector Package Diagram 255
 - Connector, Connector Package 257
 - ConnectorConstraint, Connector Package 256
 - ConnectorEnd, Connector Package 260
 - ConnectorTag, Connector Package 262
 - ConstLayoutStyles Enum 191
 - Constraint, Element Package 227
 - CreateBaselineFlag Enum 192
 - CreateModelType Enum 192
 - CustomProperties Collection, ElementFeatures Package 248
 - Diagram Package 263
 - Diagram, Diagram Package 264
 - DiagramLinks, Diagram Package 267
 - DiagramObjects, Diagram Package 268
 - Effort, Element Package 227
 - Element Package Diagram 225
 - Element Package, File 235
 - Element, Element Package 228
 - ElementFeatures Package Diagram 243
 - EmbeddedElements Collection, ElementFeatures Package 248
 - Enumerations 191
 - EnumRelationSetType Enum 192
 - Examples 181
 - Examples and Tips 184
 - ExportPackageXMIFlag Enum 193
 - Introduction 181
 - Issue, Element Package 235
 - MDGMenus Enum 193
 - Method, ElementFeatures Package 249
 - MethodConstraint, ElementFeatures Package 251
 - MethodTag, ElementFeatures Package 252
 - Metric, Element Package 236
 - Model 187
 - ObjectType Enum 193
 - Package 187
 - Parameter, ElementFeatures Package 252
 - Partitions Collection, ElementFeatures Package 253
 - Project Interface 271
 - Project, Project Interface 271
 - Properties, ElementFeatures Package 254
 - Property ElementFeatures Package 254
 - PropType Enum 194
 - Reference 186
 - ReloadType Enum 194
 - Repository 197
 - Repository Package 196, 215
 - Repository, Author Collection 210
 - Repository, Client Collection 210
 - Repository, Collection Class 211
 - Repository, Datatype 213
 - Repository, EventProperties 214
 - Repository, EventProperty 214
 - Repository, ModelWatcher 214
 - Repository, ProjectIssues 219
 - Repository, ProjectResource 220
 - Repository, PropertyType 221
 - Repository, Reference 222
 - Repository, Stereotype 223
 - Repository, Task 223
 - Repository, Term 224
 - Requirement, Element Package 237
 - Resource, Element Package 238
 - Risk, Element Package 239
 - RoleTag, Connector Package 263
 - Scenario, Element Package 239
 - ScenarioDiagramType Enum 194
 - ScenarioExtension, Element Package 240
 - ScenarioStep, Element Package 241
 - ScenarioStepType Enum 195
 - ScenarioTestType Enum 195
 - Set Up Visual Basic 183
 - Swimlane, Diagram Package 271
 - SwimlaneDef, Diagram Package 269
 - Swimlanes, Diagram Package 270
 - TaggedValue, Element Package 242
 - Test, Element Package 243
 - Transitions Collection, ElementFeatures Package 255
 - Using 181
 - VB GetObject Support 190
 - XMIType Enum 195
- Available Resources
 - Automation Interface 186

- B -

- Behavioral Model Templates 109
- Branching Macros
 - Code Template Syntax 106
- Broadcast Event
 - Add-In Model 135
 - EA_FileClose 136
 - EA_FileNew 136
 - EA_FileOpen 135
 - EA_OnPostCloseDiagram 137
 - EA_OnPostInitialized 151
 - EA_OnPostOpenDiagram 137
 - EA_OnPostTransform 151

- Broadcast Event
 - EA_OnPreExitInstance 146
 - EA_OnRetrieveModelTemplate 168
- Built-In
 - Diagram Types 53
- C -**
- Call
 - Automation Interface From Enterprise Architect 185
- CLASSGUID
 - Add-In Hidden Field 128
- CLASSTYPE
 - Add-In Hidden Field 128
- Client Collection
 - Automation Interface Repository 210
- Code Module
 - Add To MDG Technology 40
- Code Template
 - Custom Templates, Create 119
 - Default Templates 120
 - Editor 119
 - Editor, Add New Stereotyped Templates 121
 - Editor, Create Templates For Custom Languages 122
 - Editor, In SDK 118
 - Export 119
 - Framework, In SDK 88
 - Import 119
 - Syntax, Introduction 88
 - Syntax, Literal Text 88
 - Syntax, Macros 89
 - Syntax, Template Substitution Macros 89
- Code Template Syntax
 - Variable Definitions 117
 - Variable References 117
 - Variables 117
- Collection Class
 - Automation Interface Repository 211
- Collections, EASL
 - Action 111
 - Behaviors 111
 - Classifier 111
 - Construct 111
 - Node 111
 - State 111
 - State Machine 111
 - Transition 111
 - Trigger 111
 - Vertex 111
- Color Query
 - Shape Scripts 72
- COM Object
 - .NET Garbage Collect (Exit Method) 197
- Compartment Events
 - Add-In Model 158
 - EA_GetCompartmentData 159
 - EA_QueryAvailableCompartments 158
- Composite Elements
 - Metaclass, Create With Supported Attributes 23
- Concurrent Method Calls
 - In Add-Ins 126
- Conditional Substitution
 - Field Substitution Macros, Code Template Syntax 90
- Connect
 - To Automation Interface 181
- Connector
 - Add, Automation Interface Code Example 286
 - Automation Interface, Connector Package 257
 - Shape Script Properties 72
 - Tagged Value, Use 11
- Connector Package
 - Connector, Automation Interface 257
 - ConnectorConstraint, Automation Interface 256
 - ConnectorEnd, Automation Interface 260
 - ConnectorTag, Automation Interface 262
 - RoleTag, Automation Interface 263
- Connector Package Diagram
 - Automation Interface 255
- ConnectorConstraint
 - Automation Interface, Connector Package 256
- ConnectorEnd
 - Automation Interface, Connector Package 260
- ConnectorTag
 - Automation Interface, Connector Package 262
- ConstLayoutStyles Enum
 - Automation Interface 191
- Constraint
 - Automation Interface, Element Package 227
 - Profile 12
 - Stereotype 12
- Context Item Events
 - Add-In Model 155
 - EA_OnContextItemChanged 155, 157
 - EA_OnContextItemDoubleClicked 156
- Control Macros
 - Code Template Syntax 106
- Create
 - Add-In 124
 - Custom Tagged Values 86

Create
 Custom View, Add-In Model 169
 Hidden Submenu In Toolbox Profile 48
 Masked Tagged Values 86
 MDG Technologies 30
 Profiles 5
 Reference Data Tagged Values 85
 Structured Tagged Values 83
 Tasks Pane Profiles 54
 Toolbox Profile For MDG Technology 47
 UML Profiles 5

CreateBaselineFlag Enum
 Automation Interface 192

CreateModelType Enum
 Automation Interface 192

CTF
 In SDK 88

Custom
 Diagram Types 52
 Stereotypes 3

Custom Language
 Create Templates For In Code Template Editor 122

Custom Tagged Values
 Create 86

Custom View
 Add-In Model 169

CustomProperties Collection
 Automation Interface, ElementFeatures Package 248

- D -

Datatype
 Automation Interface Repository 213

Default
 Templates 120

Default Appearance
 Set For Profile Stereotype Objects 17

Default Templates
 Override in Code Template Editor 120

Default Toolbox
 Override In Profile 49

Define
 Stereotype As Metatype 21
 Stereotype Constraints 12
 Tasks Pane Contexts 57
 Tasks Pane Toolbox 55
 Validation Configuration For MDG Technology 58

Define Menu Items
 Create Add-In 124

Deploy
 Add-In 125
 MDG Technology From Add-In 59
 MDG Technology From File 59

Diagram
 Add And Manage, Automation Interface Code Example 287
 Automation Interface, Diagram Package 264
 Define Child Type, Supported Attributes 23
 Profile Attributes, PData and StyleEx 53
 Profiles 52
 Save Profile 18
 Types, Built In 53
 Types, Custom 52

Diagram Package
 Automation Interface 263
 Diagram, Automation Interface 264
 DiagramLinks, Automation Interface 267
 DiagramObjects, Automation Interface 268
 Swimlane, Automation Interface 271
 SwimlaneDef, Automation Interface 269, 270

Diagram Type
 Add To MDG Technology 36

DiagramLinks
 Automation Interface, Diagram Package 267

DiagramObjects
 Automation Interface, Diagram Package 268

Direct Substitution
 Field Substitution Macros, Code Template Syntax 90

Disable
 Add-Ins 128

Display
 Connector Properties, Shape Scripts 72
 Element Properties, Shape Scripts 72

Drawing Methods
 Shape Scripts 68

- E -

EA_Connect
 Add-In Event 130

EA_Disconnect
 Add-In Event 130

EA_FileClose
 Broadcast Events, Add-In Model 136

EA_FileNew
 Broadcast Events, Add-In Model 136

EA_FileOpen
 Broadcast Events, Add-In Model 135

EA_GetCompartmentData
 Compartment Events, Add-In Model 159

EA_GetMenuItems		
Add-In Event	131	
EA_GetMenuState		
Add-In Event	131	
EA_MenuClick		
Add-In Event	132	
EA_OnContextItemChanged		
Context Item Events, Add-In Model	155	
EA_OnContextItemDoubleClicked		
Context Item Events, Add-In Model	156	
EA_OnDeleteTechnology		
Technology Events, Add-In Model	154	
EA_OnEndValidation		
Model Validation Broadcasts, Add-In Model	161	
EA_OnImportTechnology		
Technology Events, Add-In Model	155	
EA_OnInitializeTechnologies		
Technology Events, Add-In Model	152	
EA_OnInitializeUserRules		
Model Validation Broadcasts, Add-In Model	160	
EA_OnNotifyContextItemModified		
Context Item Events, Add-In Model	157	
EA_OnOutputItemClicked		
Add-In Event	133	
EA_OnOutputItemDoubleClicked		
Add-In Event	133	
EA_OnPostActivateTechnology		
Technology Events, Add-In Model	153	
EA_OnPostCloseDiagram		
Broadcast Events, Add-In Model	137	
EA_OnPostInitialized		
Broadcast Events, Add-In Model	151	
EA_OnPostNewAttribute		
Post-New Events, Add-In Model	149	
EA_OnPostNewConnector		
Post-New Events, Add-In Model	147	
EA_OnPostNewDiagram		
Post-New Events, Add-In Model	148	
EA_OnPostNewDiagramObject		
Post-New Events, Add-In Model	148	
EA_OnPostNewElement		
Post-New Events, Add-In Model	147	
EA_OnPostNewMethod		
Post-New Events, Add-In Model	149	
EA_OnPostNewPackage		
Post-New Events, Add-In Model	150	
EA_OnPostOpenDiagram		
Broadcast Events, Add-In Model	137	
EA_OnPostTransform		
Broadcast Events, Add-In Model	151	
EA_OnPreActivateTechnology		
Technology Events, Add-In Model	152	
EA_OnPreDeleteAttribute		
Pre-Deletion Events, Add-In Model	138	
EA_OnPreDeleteConnector		
Pre-Deletion Events, Add-In Model	139	
EA_OnPreDeleteDiagram		
Pre-Deletion Events, Add-In Model	140	
EA_OnPreDeleteElement		
Pre-Deletion Events, Add-In Model	138	
EA_OnPreDeleteMethod		
Pre-Deletion Events, Add-In Model	139	
EA_OnPreDeletePackage		
Pre-Deletion Events, Add-In Model	141	
EA_OnPreDeleteTechnology		
Technology Events, Add-In Model	153	
EA_OnPreExitInstance		
Broadcast Events, Add-In Model	146	
EA_OnPreNewAttribute		
Pre-New Events, Add-In Model	144	
EA_OnPreNewConnector		
Pre-New Events, Add-In Model	142	
EA_OnPreNewDiagram		
Pre-New Events, Add-In Model	143	
EA_OnPreNewDiagramObject		
Pre-New Events, Add-In Model	143	
EA_OnPreNewElement		
Pre-New Events, Add-In Model	141	
EA_OnPreNewMethod		
Pre-New Events, Add-In Model	145	
EA_OnPreNewPackage		
Pre-New Events, Add-In Model	145	
EA_OnRetrieveModelTemplate		
Broadcast Events, Add-In Model	168	
EA_OnRunAttributeRule		
Model Validation Broadcasts, Add-In Model	163	
EA_OnRunConnectorRule		
Model Validation Broadcasts, Add-In Model	163	
EA_OnRunDiagramRule		
Model Validation Broadcasts, Add-In Model	162	
EA_OnRunElementRule		
Model Validation Broadcasts, Add-In Model	162	
EA_OnRunMethodRule		
Model Validation Broadcasts, Add-In Model	164	
EA_OnRunPackageRule		
Model Validation Broadcasts, Add-In Model	162	
EA_OnRunParameterRule		

- EA_OnRunParameterRule
 - Model Validation Broadcasts, Add-In Model 164
- EA_OnStartValidation
 - Model Validation Broadcasts, Add-In Model 161
- EA_QueryAvailableCompartments
 - Compartment Events, Add-In Model 158
- EA_ShowHelp
 - Add-In Event 134
- EAP File
 - Iterate Through, Automation Interface Code Example 284
- EASL
 - Behavioral Model Templates 109
 - Code Generation Macros, Behavioral Model 109
 - Enterprise Architect Simulation Library 109
- EASL Collections
 - Action 111
 - Behavior 111
 - Classifier 111
 - Construct 111
 - Node 111
 - State 111
 - State Machine 111
 - Transition 111
 - Trigger 111
 - Vertex 111
- EASL Properties
 - Action 113
 - Argument 113
 - Behavior 113
 - Call Event 113
 - ChangeEvent 113
 - Classifier 113
 - Condition 113
 - Construct 113
 - Edge 113
 - EventObject 113
 - Instance 113
 - Parameter 113
 - Primitive 113
 - PropertyObject 113
 - SignalEvent 113
 - State 113
 - StateMachine 113
 - TimeEvent 113
 - Transition 113
 - Trigger 113
 - Vertex 113
- EASL_GET
 - Code Generation Macro, Behavioral Model 109
- EASLList
 - Code Generation Macro, Behavioral Model 109
- Effort
 - Attributes 227
 - Automation Interface, Element Package 227
 - Methods 227
- Element
 - Add And Manage, Automation Interface Code Example 286
 - Add To Profile 6
 - Automation Interface, Element Package 228
 - Icon, Project Browser, User-Defined 19
 - Shape Script Properties 72
 - Use Extras, Automation Interface Code Example 288
- Element Package, Automation Interface
 - Constraint 227
 - Diagram 225
 - Effort 227
 - Element 228
 - File 235
 - Issue 235
 - Metric 236
 - Requirement 237
 - Resource 238
 - Risk 239
 - Scenario 239
 - ScenarioExtension 240
 - ScenarioStep 241
 - TaggedValue 242
 - Test 243
- Element Templates
 - And Profiles 3
- ElementFeatures Package, Automation Interface
 - Attribute 244
 - AttributeConstraint 246
 - AttributeTag 247
 - CustomProperties Collection 248
 - Diagram 243
 - EmbeddedElements Collection 248
 - Method 249
 - MethodConstraint 251
 - MethodTag 252
 - Parameter 252
 - Partitions Collection 253
 - Properties 254
 - Property 254
 - Transitions Collection 255
- EmbeddedElements Collection

- EmbeddedElements Collection
 - Automation Interface, ElementFeatures Package 248
 - Enable
 - Add-Ins 128
 - Enterprise Architect
 - Add-In Model 123
 - Object Model, Introduction 181
 - SDK, Introduction 2
 - Enterprise Architect Simulation Library
 - Behavioral Model Templates 109
 - EASL Code Generation 109
 - EASL_GET Macro 109
 - EASLList Macro 109
 - Enumeration
 - Automation Interface 191
 - ConstLayoutStyles 191
 - CreateBaselineFlag 192
 - CreateModelType 192
 - EnumRelationSetType 192
 - ExportPackageXMIFlag 193
 - MDGMenus 193
 - ObjectType 193
 - PropType 194
 - ReloadType 194
 - ScenarioDiagramType 194
 - ScenarioStepType 195
 - ScenarioTestType 195
 - XMIType 195
 - Enumeration Elements
 - Add To Profiles 14
 - EnumRelationSetType Enum
 - Automation Interface 192
 - EventProperties
 - Automation Interface Repository 214
 - EventProperty
 - Automation Interface Repository 214
 - Examples And Tips
 - Automation Interface 184
 - Export
 - Code Templates 119
 - Profile 17
 - UML Profile 17
 - ExportPackageXMIFlag Enum
 - Automation Interface 193
 - Extend
 - UML Toolbox Connectors 51
 - UML Toolbox Elements 50
 - Access Data from Attributes 90
 - Access Data from Classes 90
 - Access Data from Operations 90
 - Access Data from Packages 90
 - Access Data from Parameters 90
 - Conditional Substitution 90
 - Direct Substitution 90
 - File
 - Element Package, Automation Interface 235
 - Function
 - Macros, Code Template Syntax 103
- H -**
- Hidden Submenu
 - Create In Toolbox Profile 48
- I -**
- Icon
 - Project Browser, User Defined 19
 - Icons
 - For Toolbox Items, Assign 49
 - Image
 - Add To MDG Technology 43
 - Import
 - Code Templates 119
 - Instance
 - Define Behavior On Creation, Supported Attributes 22
 - Introduction
 - To Enterprise Architect SDK 2
 - To Quick Linker 25
 - To Shape Scripts 61
 - To Tagged Value Types 81
 - Issue (Defect)
 - Automation Interface, Element Package 235
 - Iterate Through EAP File
 - Automation Interface Code Example 284
- L -**
- Language
 - Custom, Create Templates For In Code Template Editor 122
 - Linked Document Template
 - Add To MDG Technology 45
 - List Macro
 - Code Template Syntax 106

- F -

Field Substitution Macros

- M -

Macro

- Branching 106
- Code Template Syntax 106
- Control 106
- Field Substitution, Code Template Syntax 90
- Function, Code Template Syntax 103
- List 106
- PI 106
- Synchronization 106
- Tagged Value, Code Template Syntax 102
- Template Substitution, Code Template Syntax 89

Macros

- Behavioral Model 109
- Code Template Syntax 89
- EASL Code Generation 109
- EASL_GET 109
- EASLList 109

Manage

- Add-Ins 128

Masked Tagged Values

- Create 86

MDA Transform

- Add To MDG Technology 42

MDG Add-Ins

- Add-In Model 170
- MDG Events 170
- MDG_BuildProject 171
- MDG_Connect 171
- MDG_Disconnect 172
- MDG_GetConnectedPackages 173
- MDG_GetProperty 173
- MDG_Merge 174
- MDG_NewClass 175
- MDG_PostGenerate 176
- MDG_PostMerge 176
- MDG_PreGenerate 177
- MDG_PreMerge 177
- MDG_PreReverse 178
- MDG_Run_Exe 179
- MDG_View 179

MDG Events

- Add-In Model 170
- MDG_BuildProject 171
- MDG_Connect 171
- MDG_Disconnect 172
- MDG_GetConnectedPackages 173
- MDG_GetProperty 173
- MDG_Merge 174

- MDG_NewClass 175
- MDG_PostGenerate 176
- MDG_PostMerge 176
- MDG_PreGenerate 177
- MDG_PreMerge 177
- MDG_PreReverse 178
- MDG_Run_Exe 179
- MDG_View 179

MDG Technology

- Create 30
- Create Toolbox Profile For 47
- Define Tasks Pane Profile 54
- Define Validation Configuration 58
- Deploy From Add-In 59
- Deploy From File 59
- Example Of Development 30
- In SDK 30
- Include Custom Diagram Types 52
- Incorporate Model Template 59
- MDG Technology Selection (MTS) File 46
- MDG Technology Wizard
 - Add Code Modules 40
 - Add Diagram Type To Technology 36
 - Add Images 43
 - Add Linked Document Template To Technology 45
 - Add MDA Transforms 42
 - Add Pattern To Technology 35
 - Add Profile To Technology 35
 - Add RTF Report Template To Technology 44
 - Add Scripts 43
 - Add Tagged Value Types 39
 - Add Task Panel To Technology 38
 - Add Toolbox To Technology 37
 - Create Technologies 30

MDG_BuildProject

- Add-In Model 171

MDG_Connect

- Add-In Model 171

MDG_Disconnect

- Add-In Model 172

MDG_GetConnectedPackages

- Add-In Model 173

MDG_GetProperty

- Add-In Model 173

MDG_Merge

- Add-In Model 174

MDG_NewClass

- Add-In Model 175

MDG_PostGenerate

- Add-In Model 176

MDG_PostMerge

- MDG_PostMerge
 - Add-In Model 176
 - MDG_PreGenerate
 - Add-In Model 177
 - MDG_PreMerge
 - Add-In Model 177
 - MDG_PreReverse
 - Add-In Model 178
 - MDG_Run_Exec
 - Add-In Model 179
 - MDG_View
 - Add-In Model 179
 - MDGMenus Enum
 - Automation Interface 193
 - Menu
 - Items, Define In Add-In 124
 - Metaclass
 - Add To Profile 6
 - Method
 - Add And Delete, Automation Interface Code Example 288
 - Automation Interface, ElementFeatures Package 249
 - Work With, Automation Interface Code Example 293
 - MethodConstraint
 - Automation Interface, ElementFeatures Package 251
 - MethodTag
 - Automation Interface, ElementFeatures Package 252
 - Metric
 - Automation Interface, Element Package 236
 - MigrateToBPMN11()
 - Function 271
 - MiscData 228
 - Model
 - Automation Interface 187
 - Templates, Incorporate In Technology 59
 - Model Search
 - Access From Add-In 128
 - Define In MTS File 46
 - Model Validation
 - Define Configuration For MDG Technology 58
 - Model Validation Broadcasts
 - Add-In Model 160
 - EA_OnEndValidation 161
 - EA_OnInitializeUserRules 160
 - EA_OnRunAttributeRule 163
 - EA_OnRunConnectorRule 163
 - EA_OnRunDiagramRule 162
 - EA_OnRunElementRule 162
 - EA_OnRunMethodRule 164
 - EA_OnRunPackageRule 162
 - EA_OnRunParameterRule 164
 - EA_OnStartValidation 161
 - Model Validation Example 165
 - Model Views
 - Define In MTS File 46
 - Models Collection 197
 - ModelWatcher
 - Automation Interface Repository 214
 - MTS File
 - Advanced Options 46
 - Create 46
 - Incorporate Model Search 46
 - Incorporate Model View 46
 - Working With 46
 - Multiple Stereotype
 - Restrict Application Of 21
- O -**
- ObjectType Enum
 - Automation Interface 193
 - Open Repository
 - Automation Interface Code Example 284
 - Override
 - Default Toolbox In Toolbox Profile 49
- P -**
- Package
 - Add And Manage, Automation Interface Code Example 285
 - Automation Interface 187
 - Automation Interface Repository 215
 - Profile 5
 - Save Profile 18
 - Parameter
 - Automation Interface, ElementFeatures Package 252
 - Partitions Collection
 - Automation Interface, ElementFeatures Package 253
 - Pattern
 - Add To MDG Technology 35
 - PDATA
 - Diagram Profile Attribute Values 53
 - Element Attribute In MiscData, Object Model 228
 - PI Macro
 - Code Template Syntax 106
 - Post-New Events
 - Add-In Model 146

- Post-New Events
 - EA_OnPostNewAttribute 149
 - EA_OnPostNewConnector 147
 - EA_OnPostNewDiagram 148
 - EA_OnPostNewDiagramObject 148
 - EA_OnPostNewElement 147
 - EA_OnPostNewMethod 149
 - EA_OnPostNewPackage 150
 - Predefined Tag Type
 - Assign To Stereotype 9
 - Define 9
 - Predefined Tagged Value Type
 - Filters 81
 - Reference Data 84
 - Structured 81
 - Syntax 81, 84
 - Pre-Deletion Events
 - Add-In Model 138
 - EA_OnPreDeleteAttribute 138
 - EA_OnPreDeleteConnector 139
 - EA_OnPreDeleteDiagram 140
 - EA_OnPreDeleteElement 138
 - EA_OnPreDeleteMethod 139
 - EA_OnPreDeletePackage 141
 - Pre-New Events
 - Add-In Model 141
 - EA_OnPreNewAttribute 144
 - EA_OnPreNewConnector 142
 - EA_OnPreNewDiagram 143
 - EA_OnPreNewDiagramObject 143
 - EA_OnPreNewElement 141
 - EA_OnPreNewMethod 145
 - EA_OnPreNewPackage 145
 - Profile
 - Add Elements 6
 - Add Enumeration Elements 14
 - Add Metaclasses 6
 - Add Shape Script 15
 - Add Stereotypes 6
 - Add To MDG Technology 35
 - And Element Templates 3
 - Constraints 12
 - Create 5
 - Diagram, Create 52
 - Export 17
 - Import From XML 3
 - Package 5
 - Save From Diagram 18
 - Save From Package 18
 - Set Default Appearance Of Stereotype Objects 17
 - Stereotype 5, 24
 - Stereotypes 3
 - Tags 8
 - Tasks Pane, Create 54
 - Toolbox 47
 - Work With 5
 - Project Browser
 - Icon, User-Defined 19
 - Project Interface
 - Automation Interface 271
 - Project 271
 - ProjectIssues
 - Automation Interface Repository 219
 - ProjectResource
 - Automation Interface Repository 220
 - Properties
 - Automation Interface, ElementFeatures Package 254
 - Properties, EASL
 - Action 113
 - Argument 113
 - Behavior 113
 - Call Event 113
 - ChangeEvent 113
 - Classifier 113
 - Condition 113
 - Construct 113
 - Edge 113
 - EventObject 113
 - Instance 113
 - Parameter 113
 - Primitive 113
 - PropertyObject 113
 - SignalEvent 113
 - State 113
 - StateMachine 113
 - TimeEvent 113
 - Transition 113
 - Trigger 113
 - Vertex 113
 - Property
 - Automation Interface, ElementFeatures Package 254
 - PropertyType
 - Automation Interface Repository 221
 - PropType Enum
 - Automation Interface 194
- Q -**
- Query Methods
 - In Shape Scripts 72
 - Quick Linker

Quick Linker

- Connector Names 29
- Default Settings, Hide 28
- Definition Format 25
- Element Names 29
- Example 27
- Introduction 25

- R -

Re-entrancy

- In Add-Ins 126

Reference

- Automation Interface 186
- Automation Interface Repository 222

Reference Data Tagged Value Type 84

Reference Data Tagged Values

- Create 85

ReloadType Enumeration

- Automation Interface 194

Repository

- Attributes 197
- Author Collection 210
- Automation Interface 197
- Client Collection 210
- Collection Class 211
- Datatype 213
- EventProperties 214
- EventProperty 214
- Methods 197
- ModelWatcher 214
- Open, Automation Interface Code Example 284
- Package 215
- Package, Automation Interface 196
- ProjectIssues 219
- ProjectResource 220
- PropertyType 221
- Reference 222
- Stereotype 223
- Task 223
- Term 224
- Use Extras, Automation Interface Code Example 290

Requirement

- Automation Interface, Element Package 237

Reserved Names

- In Shape Scripts, Connectors 76
- In Shape Scripts, Elements 76

Resource

- Automation Interface, Element Package 238

Risk

- Automation Interface, Element Package 239

RoleTag

- Automation Interface, Connector Package 263

RTF Report Template

- Add To MDG Technology 44

- S -

Save

- Profile From Diagram Context 18
- Profile From Package Context 18
- Tasks Pane Profile 58

Scenario

- Automation Interface, Element Package 239

ScenarioDiagramType Enum

- Automation Interface 194

ScenarioExtension

- Automation Interface, Element Package 240

ScenarioStep

- Automation Interface, Element Package 241

ScenarioStepType Enum

- Automation Interface 195

ScenarioTestType Enum

- Automation Interface 195

Script

- Add To MDG Technology 43

SDK

- Enterprise Architect 2

Search

- Add-In 128

Search Data Parameter

- Add-In Search 129

Shape

- <LabelID> 76
- Attributes 66
- Decoration 76
- Editor 64
- Label 76
- Main 76
- Source 76
- Target 76

Shape Attributes

- Shape Scripts 66

Shape Editor 64

Shape Scripts

- Add To Profile 15
- Arithmetical Operations 76
- Assign To Stereotype 61
- Basic Shapes 77
- Change Font Of Text 76
- Cloud Path 77

- Shape Scripts
 - Color Queries 72
 - Comments 76
 - Conditional Branching 72
 - Connector 77
 - Create 61
 - Custom Shapes 61
 - Display Element Properties 72
 - Double Line 77
 - Drawing Methods 68
 - Editable Field 77
 - Example Shape Scripts 77
 - Filled Arrow 77
 - Fonts 76
 - Getting Started 61
 - Introduction 61
 - Looping 76
 - Miscellaneous 76
 - Multiple Condition 77
 - Override Element Appearance 61
 - Properties, Connector 72
 - Properties, Element 72
 - Query Methods 72
 - Reserved Names, Connectors 76
 - Reserved Names, Elements 76
 - Return Statement 77
 - Shape Attributes 66
 - Shape Editor 64
 - Single Condition 77
 - Stereotypes 61
 - String Manipulation 76
 - Subshape 77
 - Subshape Layout 75
 - Syntax Grammar 65
 - Terminate Execution 76
 - Variable Declarations 76
 - Without Stereotypes 76
 - Writing Scripts 65
 - Software Development Kit
 - Enterprise Architect 2
 - Stereotype
 - Add Shape Script In Profile 15
 - Add To Profile 6
 - Add, Automation Interface Code Example 292
 - Automation Interface Repository 223
 - Custom 3
 - Define As Metatype 21
 - Dialog 3
 - Multiple, Restrict Application Of 21
 - Predefined Tag Types 9
 - Profile 24
 - Set Default Appearance Of Objects In Profile 17
 - Tagged Values In Profile 8
 - Tags For Supported Attributes 10
 - Tags, Define 8
 - Structured Tagged Value Type 81
 - Structured Tagged Values
 - Create 83
 - StyleEx
 - Diagram Profile Attribute Values 53
 - Submenu
 - Hidden, Create In Toolbox Profile 48
 - Subshape
 - Example 75
 - In Shape Scripts 75
 - Substitution
 - Conditional 90
 - Direct 90
 - Macro 72
 - Supported Attribute
 - Create Composite Elements 23
 - Define Behavior On Creating Instance 22
 - Define Child Diagram Types 23
 - Metatype, In UML Profiles 19
 - Of Stereotype Tags 10
 - Stereotype, In UML Profiles 19
 - Supported Stereotype Attribute Tags 10
 - SwimlaneDef
 - Automation Interface, Diagram Package 269
 - Swimlanes
 - Automation Interface, Diagram Package 270, 271
 - Synchronization
 - Macros, Code Template Syntax 106
 - Syntax Grammar
 - Shape Scripts 65
- T -
- Tag
 - Profile 8
 - Tag Type
 - Predefined, Assign To Stereotype 9
 - Tagged Value
 - Connector, Use 11
 - Custom, Create 86
 - Element Package, Automation Interface 242
 - Macros, Code Template Syntax 102
 - Masked, Create 86
 - Reference Data, Create 85
 - Structured, Create 83
 - Tagged Value Type

- Tagged Value Type
 - Add To MDG Technology 39
 - Filters 81
 - Introduction 81
 - Predefined Reference Data 84
 - Predefined Structured 81
 - TaggedValue
 - Automation Interface, Element Package 242
 - Task
 - Automation Interface Repository 223
 - Task Panel
 - Add To MDG Technology 38
 - Tasks Pane
 - Allocate Toolbox To Contexts 57
 - Commands, Built In 55
 - Contexts, Define 57
 - Named Contexts 57
 - Profiles, Create 54
 - Run Add-In Functions From 56
 - Save Profile 58
 - Toolboxes, Define 55
 - Technology Event
 - EA_OnInitializeTechnologies 152
 - Technology Events
 - Add-In Model 152
 - EA_OnDeleteTechnology 154
 - EA_OnImportTechnology 155
 - EA_OnPostActivateTechnology 153
 - EA_OnPreActivateTechnology 152
 - EA_OnPreDeleteTechnology 153
 - Template
 - Behavioral Model 109
 - Editor In SDK 118
 - Model, Incorporate In Technology 59
 - Term
 - Automation Interface Repository 224
 - Test
 - Automation Interface, Element Package 243
 - Toolbox
 - Add To MDG Technology 37
 - Connectors For Extending In Profile 51
 - Customize 47
 - Default, Override In Profile 49
 - Elements For Extending 50
 - Override Default In Toolbox Profile 49
 - Page Attributes 48
 - Profile, Create For MDG Technology 47
 - Profiles 47
 - Tasks Pane, Define 55
 - Toolbox Profile
 - Connectors For Extending 51
 - Create Hidden Submenu In 48
 - Items, Assign Icons For 49
 - Pages That Can be Overridden 50
 - Transitions Collection
 - Automation Interface, ElementFeatures Package 255
 - Tricks and Traps
 - Create Add-In 126
- U -**
- UML Profile
 - And Element Templates 3
 - Create 5
 - Export 17
 - Import From XML 3
 - Save From Diagram 18
 - Save From Package 18
 - Stereotypes 3
 - Work With 5
 - Use Element Extras
 - Automation Interface Code Example 288
 - Use Repository Extras
 - Automation Interface Code Example 290
- V -**
- Validation
 - Of Model, Configure For MDG Technology 58
 - Variable
 - Definitions, Code Template Syntax 117
 - Definitions, Examples 117
 - References, Code Template Syntax 117
 - References, Examples 117
 - VB
 - Set Up In Automation Interface 183
 - Visual Basic
 - Connect To Automation Interface 181
 - Set Up In Automation Interface 183
- W -**
- Work With
 - Attributes, Automation Interface Code Example 292
 - Methods, Automation Interface Code Example 293
- X -**
- XMIType Enum
 - Automation Interface 195

Enterprise Architect Software Developers' Kit

www.sparxsystems.com