



# Enterprise Architect

User Guide Series

# Enterprise Architect Add-In Model

Author: Sparx Systems

Date: 15/07/2016

Version: 1.0

# Table of Contents

Enterprise Architect Add-In Model .....	5
The Add-In Manager .....	6
Add-In Tasks .....	7
Create Add-Ins .....	8
Define Menu Items .....	9
Deploy Add-Ins .....	11
Tricks and Traps .....	13
Add-In Search .....	15
XML Format (Search Data) .....	16
Add-In Events .....	18
EA_Connect .....	19
EA_Disconnect .....	20
EA_GetMenuItems .....	21
EA_GetMenuState .....	22
EA_GetRibbonCategory .....	24
EA_MenuClick .....	26
EA_OnOutputItemClicked .....	28
EA_OnOutputItemDoubleClicked .....	29
EA_ShowHelp .....	30
Broadcast Events .....	31
Schema Composer Broadcasts .....	33
EA_GenerateFromSchema .....	34
EA_GetProfileInfo .....	35
EA_IsSchemaExporter .....	36
Add-In License Management Events .....	37
EA_AddinLicenseValidate .....	38
EA_AddinLicenseGetDescription .....	39
EA_GetSharedAddinName .....	40
Compartment Events .....	42
EA_QueryAvailableCompartments .....	43
EA_GetCompartmentData .....	45
Context Item Events .....	48
EA_OnContextItemChanged .....	49
EA_OnContextItemDoubleClicked .....	50
EA_OnNotifyContextItemModified .....	51
EA_FileClose .....	52
EA_FileNew .....	53
EA_FileOpen .....	54
EA_OnPostCloseDiagram .....	55
EA_OnPostInitialized .....	56
EA_OnPostOpenDiagram .....	57
EA_OnPostTransform .....	58
EA_OnPreExitInstance .....	59
EA_OnRetrieveModelTemplate .....	60
EA_OnTabChanged .....	62
Model Validation Broadcasts .....	63
EA_OnInitializeUserRules .....	64

EA_OnStartValidation .....	65
EA_OnEndValidation .....	66
EA_OnRunElementRule .....	67
EA_OnRunPackageRule .....	68
EA_OnRunDiagramRule .....	69
EA_OnRunConnectorRule .....	70
EA_OnRunAttributeRule .....	71
EA_OnRunMethodRule .....	72
EA_OnRunParameterRule .....	73
Model Validation Example .....	74
Post-New Events .....	80
EA_OnPostNewElement .....	81
EA_OnPostNewConnector .....	82
EA_OnPostNewDiagram .....	83
EA_OnPostNewDiagramObject .....	84
EA_OnPostNewAttribute .....	85
EA_OnPostNewMethod .....	86
EA_OnPostNewPackage .....	87
EA_OnPostNewGlossaryTerm .....	88
Pre-Deletion Events .....	89
EA_OnPreDeleteElement .....	90
EA_OnPreDeleteAttribute .....	91
EA_OnPreDeleteMethod .....	92
EA_OnPreDeleteConnector .....	93
EA_OnPreDeleteDiagram .....	94
EA_OnPreDeleteDiagramObject .....	95
EA_OnPreDeletePackage .....	96
EA_OnPreDeleteGlossaryTerm .....	97
Pre New-Object Events .....	98
EA_OnPreNewElement .....	99
EA_OnPreNewConnector .....	100
EA_OnPreNewDiagram .....	101
EA_OnPreNewDiagramObject .....	102
EA_OnPreDropFromTree .....	103
EA_OnPreNewAttribute .....	104
EA_OnPreNewMethod .....	105
EA_OnPreNewPackage .....	106
EA_OnPreNewGlossaryTerm .....	107
Tagged Value Broadcasts .....	108
EA_OnAttributeTagEdit .....	109
EA_OnConnectorTagEdit .....	110
EA_OnElementTagEdit .....	111
EA_OnMethodTagEdit .....	112
Technology Events .....	113
EA_OnInitializeTechnologies .....	114
EA_OnPreActivateTechnology .....	115
EA_OnPostActivateTechnology .....	116
EA_OnPreDeleteTechnology .....	117
EA_OnDeleteTechnology .....	118
EA_OnImportTechnology .....	119
Custom Views .....	120

Create a Custom View .....	121
Add a Portal .....	122
Custom Docked Window .....	123
MDG Add-Ins .....	125
MDG Events .....	126
MDG_Build Project .....	127
MDG_Connect .....	128
MDG_Disconnect .....	129
MDG_GetConnectedPackages .....	130
MDG_GetProperty .....	131
MDG_Merge .....	132
MDG_NewClass .....	135
MDG_PostGenerate .....	136
MDG_PostMerge .....	137
MDG_PreGenerate .....	138
MDG_PreMerge .....	139
MDG_PreReverse .....	140
MDG_RunExe .....	141
MDG_View .....	142

# Enterprise Architect Add-In Model



The **Add-In** facility provides a means of extending Enterprise Architect, allowing the programmer to enhance the user interface by adding new menus, sub menus, windows and other controls to perform a variety of functions. An Add-In is an ActiveX COM object that is notified of events in the user interface, such as mouse clicks and element selections, and has access to the repository content through the **Object Model**. Add-Ins can also be integrated with the license management system.

Using this powerful facility, you can extend Enterprise Architect to create new features not available in the core product, and these can be compiled and easily distributed to a community of users within an organization, or more broadly to an entire industry. Using the Add-In facility it is even possible to create support for modeling languages and frameworks not supported in the core product.

Add-Ins have several advantages over stand-alone automation clients:

- 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 (Add-Ins created before 2004 are no longer supported - if an Add-In subscribes to the Addn\_Tmpl.tlb interface (2003 style), it fails 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, UML Profiles and the **Automation Interface**. Once an Add-In is registered, it can be managed using the Add-In Manager.

# The Add-In Manager

If you want to check what **Add-Ins** are available on your system, and enable or disable them for use, you can review the '**Add-In Manager**' dialog. This dialog lists the Add-Ins that have been registered on your system, and their current status (Enabled or Disabled).

## Access

Ribbon	Extend > Configure > Manage <b>Add-Ins</b>
Menu	Extensions   Manage <b>Add-Ins</b>

## Enable/disable Add-Ins

Action	Detail
Enable an Add-In	<p>To enable an <b>Add-In</b> so that it is available for use, select the 'Load on Startup' check box to the left of the name.</p> <p>Click on the <b>OK button</b>.</p> <ul style="list-style-type: none"><li>Any Add-In specific features, facilities and Help are made available through the 'Extensions   &lt;add-in name&gt;' menu option</li><li>Any defined Add-In windows are populated with information; select 'Extensions   Add-In Windows'</li></ul>
Disable an Add-In	<p>To disable an <b>Add-In</b> so that it is not available for use, clear the 'Load on Startup' check box against the name.</p> <p>Click on the <b>OK button</b>.</p> <p>All menu options, features and facilities specific to the Add-In are hidden and made inactive.</p>

## Notes

- When you enable or disable an **Add-In**, you must re-start Enterprise Architect to action the change

# Add-In Tasks

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

## Create an Add-In

Topic
Create an <b>Add-In</b> .
Define Menu Items.
Respond to Menu Events.
Handle <b>Add-In</b> Events.

## Deploy your Add-In

Topic
Potential Pitfalls.

## Manage Add-Ins

Topic
Register an <b>Add-In</b> (developed in-house or brought-in).
The <b>Add-In</b> Manager.

## 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. To help with this, you could review some examples of Automation Interfaces - examples of code used to create **Add-Ins** for Enterprise Architect - on the Sparx Systems web page.

### Create an Enterprise Architect Add-In

Step	Action
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.
3	Create a COM Class and implement each of the general <b>Add-In</b> Events applicable to your Add-In. You only have to define methods for events to respond to.
4	Add a registry key that identifies your <b>Add-In</b> to Enterprise Architect, as described in the Deploy <b>Add-Ins</b> topic.



# Define Menu Items

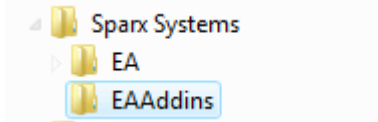
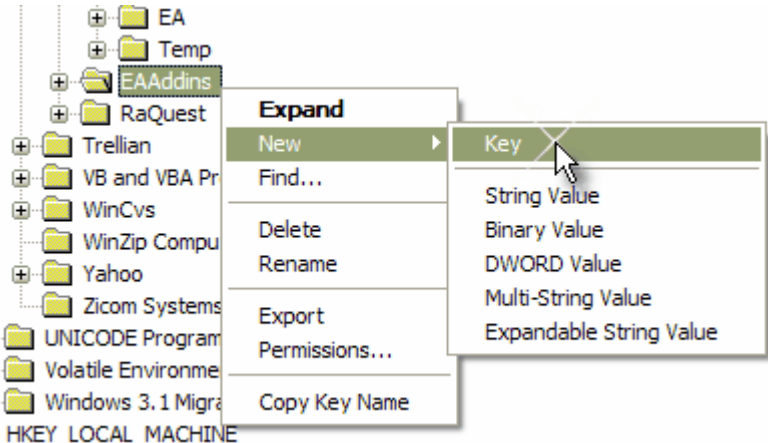
## Tasks

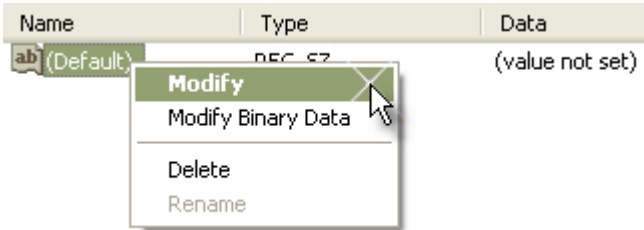
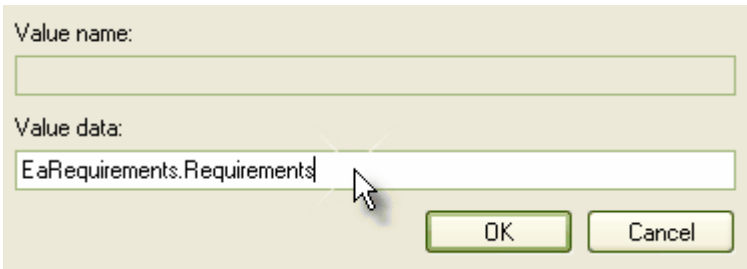
Task	Detail
Define Menu Items	<p>Menu items are defined by responding to the GetMenuItems event.</p> <p>The first time this event is called, MenuName is an empty string, representing the top-level menu. For a simple <b>Add-In</b> with just a single menu option you can return a string.</p> <pre> Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant     EA_GetMenuItems = "&amp;Joe's Add-In" End Function </pre>
Define Sub-Menus	<p>To define sub-menus, prefix a parent menu with a dash. Parent and sub-items are defined like this:</p> <pre> Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant     Select Case MenuName     Case ""         'Parent Menu Item         EA_GetMenuItems = "-&amp;Joe's <b>Add-In</b>"     Case "-&amp;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("&amp;Diagram", "&amp;Treeview")     Case Else         MsgBox "Invalid Menu", vbCritical     End Select End Function </pre>
Define Further Sub-Menus	<p>Similarly, you can define further sub-items:</p> <pre> Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant     Select Case MenuName     Case ""         EA_GetMenuItems = "-Joe's <b>Add-In</b>"     Case "-Joe's Add-In"         EA_GetMenuItems = Array("-&amp;Diagram", "&amp;TreeView")     Case "-&amp;Diagram"         EA_GetMenuItems = "&amp;Properties"     Case Else         MsgBox "Invalid Menu", vbCritical     End Select </pre>

	End Function
Enable/Disable menu options	<p>To enable or disable menu options by default, you can use this method to show particular items to the user:</p> <pre>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 "&amp;Translate", "Save &amp;Project"             If GetIsProjectSelected() Then                 IsEnabled = <b>False</b>             End If         End Select     End Select     IsChecked = GetIsCurrentSelection() End Sub</pre>

# Deploy Add-Ins

## Deploy Add-Ins to users' sites

Step	Action
1	Add the <b>Add-In</b> 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: <ul style="list-style-type: none"> <li>• If compiled as a native Win32 DDL, such as VB6 or C++, register the DDL using the regsvr32 command from the command prompt regsvr32 "C:\Program Files\MyCompany\EAAddin\EAAddin.dll"</li> <li>• If compiled as a .NET DLL, such as C# or VB.NET, register the DLL using the RegAsm command from the command prompt C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe "C:\Program Files\MyCompany\EAAddin\EAAddin.dll"/codebase</li> </ul>
3	Place a new entry into the registry using the registry editor (run regedit) so that Enterprise Architect recognizes the presence of your <b>Add-In</b> .
4	Add a new key value EAAddIns under the location: <ul style="list-style-type: none"> <li>• HKEY_CURRENT_USER\Software\Sparx Systems for single users</li> <li>• HKEY_LOCAL_MACHINE\Software\Sparx Systems for multiple users on a machine</li> </ul> 
5	Add a new key under this key with the project name.  <p>(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</p>

	the project file.
6	<p>Specify the default value by modifying the default value of the key.</p> 
7	<p>Enter the value of the key by typing in the (project name).(class name), such as: EaRequirements.Requirements where <i>EaRequirements</i> is the project name, as shown in this example:</p> 

# Tricks and Traps

## Considerations

Item	Detail
Visual Basic 5/6 Users Note	<p>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 this:</p> <pre>Reference=*\G{64FB2BF4-9EFA-11D2-8307-C45586000000}#2.2#0#...\Program Files\Sparx Systems\EA\EA.TLB#Enterprise Architect <b>Object Model 2.02</b></pre> <p>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.</p>
Add-In Fails to Load	<p>From Enterprise Architect release 7.0, <b>Add-Ins</b> created before 2004 are no longer supported. If an <b>Add-In</b> subscribes to the Addn_Tmpl.tlb interface (2003 style), it fails on load. In this event, contact the vendor or author of the Add-In and request an upgrade.</p>
Holding State Information	<p>It is possible for an <b>Add-In</b> 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:</p> <ul style="list-style-type: none"> <li>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</li> <li>When you close Enterprise Architect, all <b>Add-Ins</b> 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</li> <li>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)</li> </ul> <p>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.</p>
Enterprise Architect Not Closing	<p>.NET Specific Issues</p> <p>Automation checks the use of objects and won't allow any of them to be destroyed until they are no longer being used.</p> <p>As noted in the <b>Automation Interface</b> 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:</p> <pre>GC.Collect(); GC.WaitForPendingFinalizers();</pre> <p>Additionally, because automation clients hook into Enterprise Architect, which creates <b>Add-Ins</b> 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 <b>Add-In</b> might retain hooks into</p>

	<p>Enterprise Architect because:</p> <ul style="list-style-type: none"> <li>• It keeps a private reference to an Enterprise Architect object (see <i>Holding State Information</i> above), or</li> <li>• It has been created by .NET and the GC mechanism hasn't got around to releasing it</li> </ul> <p>There are two actions required to avoid deadlock situations:</p> <ul style="list-style-type: none"> <li>• Automation controllers must call Repository.CloseAddins() at some point (presumably at the end of processing)</li> <li>• Add-Ins must release all references to Enterprise Architect in the Disconnect() event; see the <i>Add-In Events</i> topic for details</li> </ul> <p>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.</p> <p><b>Miscellaneous</b></p> <p>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.</p> <p>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.</p> <p>You can use your private Add-In key (as required for Add-In deployment) to store configuration information pertinent to your Add-In.</p>
Concurrent Calls	<p>In Enterprise Architect releases up to release 7.0, there is a possibility that Enterprise Architect could call two <b>Add-In</b> methods concurrently if the Add-In calls:</p> <ul style="list-style-type: none"> <li>• A message box</li> <li>• A modal dialog</li> <li>• VB DoEvents, .NET Application DoEvents or the equivalent in other languages</li> </ul> <p>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.</p> <p>If developing <b>Add-Ins</b>, 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.</p>

## Add-In Search

Enterprise Architect enables Extensions to integrate with the **Model Search**. 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 like this:

Function <method name> (ByVal Rep As Repository, ByVal SearchText As String, ByRef XMLResults As String) As Variant

- Rep - EA.Repository - IN - The current open repository
- SearchText - String - IN - An optional field that you can fill in through the Model Search
- XMLResults - String - OUT - 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

### Return Value

The method must return any non-empty 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 in which 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

## XML Format (Search Data)

This example XML provides the format for the sSearchData parameter of the RunModelSearch method.

```
<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 here.
```

```
-->
```

```
<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>

# Add-In Events

All Enterprise Architect **Add-Ins** can choose to respond to general **Add-In** events.

## Events

Event
<i>EA_Connect</i> - <b>Add-Ins</b> can use this to identify their type and to respond to Enterprise Architect start up.
<i>EA_Disconnect</i> - <b>Add-Ins</b> can use this to respond to user requests to disconnect the model branch from an external project.
<i>EA_GetMenuItems</i> - <b>Add-Ins</b> can use this to provide the Enterprise Architect user interface with additional <b>Add-In</b> menu options in various context and main menus.
<i>EA_GetMenuState</i> - <b>Add-Ins</b> can use this to set a particular menu option to either enabled or disabled.
<i>EA_GetRibbonCategory</i> - <b>Add-Ins</b> can use this to identify the Ribbon panel in which to house their calling icon.
<i>EA_MenuClick</i> - received by an <b>Add-In</b> in response to user selection of a menu option.
<i>EA_OnOutputItemClicked</i> - informs <b>Add-Ins</b> that the user has clicked on a list entry in the system tab or one of the user defined output tabs.
<i>EA_OnOutputItemDoubleClicked</i> - informs <b>Add-Ins</b> that the user has used the mouse to double-click on a list entry in one of the user-defined output tabs.
<i>EA_ShowHelp</i> - <b>Add-Ins</b> can use this to show a Help topic for a particular menu option.

## EA\_Connect

**Add-Ins** can use EA\_Connect events 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.

### Syntax

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

The EA\_Connect function syntax has this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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

A string identifying a specialized type of **Add-In**:

Type	Details
"MDG"	MDG <b>Add-Ins</b> receive MDG Events and extra menu options.
""	A non-specialized <b>Add-In</b> .

## EA\_Disconnect

**Add-Ins** can use the EA\_Disconnect event to respond to user requests to disconnect the model branch from an external project.

This function is called when 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:

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

### Syntax

```
Sub EA_Disconnect()
```

### Return Value

None.

## EA\_GetMenuItems

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* topic.

### Syntax

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

The EA\_GetMenuItems function syntax has these parameters.

Parameter	Type
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the menu. This can be TreeView, MainMenu or Diagram.
MenuName	String Direction: IN Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.
Repository	EA.Repository Direction: IN Description: 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 these 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.

## EA\_GetMenuState

**Add-Ins** can use the EA\_GetMenuState event 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 further described in the *Define Menu Items* topic.

### 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 these parameters.

Parameter	Type
IsChecked	Boolean Direction: OUT Description: Set to <b>True</b> to check this particular menu option.
IsEnabled	Boolean Direction: OUT Description: Set to <b>False</b> to disable this particular menu option.
ItemName	String Direction: IN Description: The name of the option actually clicked; for example, 'Create a New Invoice'.
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the menu. This can be TreeView, MainMenu or Diagram.
MenuName	String Direction: IN Description: 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 Direction: IN Description: 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.

## EA\_GetRibbonCategory

**Add-Ins** can use EA\_GetRibbonCategory events to identify the Ribbon in which the **Add-In** should place its menu icon.

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 use for EA\_GetRibbonCategory is in initializing the Add-In access point.

### Syntax

Function EA\_GetRibbonCategory (Repository As EA.Repository) As String

The EA\_GetRibbonCategory function syntax has this parameter:

Parameter	Description
Repository	EA.Repository Direction: IN Description: 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

A string matching the name of the selected ribbon (in English if you are using a translated version). The possible names are:

- Start
- Design
- Layout
- Publish
- Configure
- Construct
- Code
- Simulate
- Execute
- Extend

It is not possible to include **Add-Ins** in the Specification - Specify ribbon or Documentation - Edit ribbon.

If the function isn't implemented (or if an invalid name is returned) the **Add-In** menu will be available from the Extend ribbon, Add-Ins panel.

### Learn more

- [MDG Add-Ins](#)
- [Using the Configure Panel](#)

-





## EA\_MenuClick

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:

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

This code is 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 methods.

### Syntax

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

The EA\_GetMenuClick function syntax has these parameters.

Parameter	Type
ItemName	String Direction: IN Description: The name of the option actually clicked; for example, 'Create a New Invoice'.
MenuLocation	String Direction: IN Description: A string representing the part of the user interface that brought up the menu. This can be TreeView, MainMenu or Diagram.
MenuName	String Direction: IN Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnOutputItemClicked

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.

### Syntax

EA\_OnOutputItemClicked (Repository As EA.Repository, TabName As String, LineText As String, ID As Long)

The EA\_OnOutputItemClicked function syntax has these parameters.

Parameter	Type
ID	Long Direction: IN Description: The ID value specified in the original call to Repository.WriteOutput().
LineText	String Direction: IN Description: The text that had been supplied as the String parameter in the original call to 'Repository.WriteOutput()'.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab that the click occurred in. Usually this would have been created through 'Repository.AddTab()'.

### Return Value

None.

## EA\_OnOutputItemDoubleClicked

EA\_OnOutputItemDoubleClicked events inform **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.

### Syntax

EA\_OnOutputItemDoubleClicked (Repository As EA.Repository, TabName As String, LineText As String, ID As Long)

The EA\_OnOutputItemClicked function syntax contains these parameters.

Parameter	Type
ID	Long Direction: IN Description: The ID value specified in the original call to Repository.WriteOutput().
LineText	String Direction: IN Description: The text that had been supplied as the String parameter in the original call to 'Repository.WriteOutput()'.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model; poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab that the click occurred in; usually this would have been created through 'Repository.AddTab()'.

### Return Value

None.

## EA\_ShowHelp

**Add-Ins** can use the EA\_ShowHelp event to show a Help topic for a particular menu option. When the user has an **Add-In** menu option selected, pressing **F1** can be related 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.

### Syntax

Sub EA\_ShowHelp (Repository as EA.Repository, MenuLocation As String, MenuName as String, ItemName as String)

The EA\_ShowHelp function syntax contains these parameters.

Parameter	Type
ItemName	String Direction: Description: The name of the option actually clicked; for example, 'Create a New Invoice'.
MenuLocation	String Direction: Description: A string representing the part of the user interface that brought up the menu. This can be Treeview, MainMenu or Diagram.
MenuName	String Direction: Description: The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu this is an empty string.
Repository	EA.Repository Direction: IN Description: 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.

# Broadcast Events

## Overview

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.

MDG Events add 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.

Event Type
<b>Add-In</b> Licence Management Events
Compartment Events
Context Item Events
File Close Event
File New Event
File Open Event
<b>Model Validation</b> Broadcasts
On Tab Changed Event
Post Close Diagram Event
Post Initialization Event
Post New Events
Post Open Diagram Event
Pre-Deletion Events
Pre-Exit Instance (not currently used)
On the creation of new objects
Retrieve Model Template Event
<b>Schema Composer</b> Broadcasts
<b>Tagged Value</b> Broadcasts
Technology Events

Transformation Event



## Schema Composer Broadcasts

Enterprise Architect **Add-Ins** can respond to events associated with the **Schema Composer** to provide custom schema export formats.

The requirements for an **Add-In** to participate consist of implementing these three functions:

- EA\_IsSchemaExporter
- EA\_GetProfileInfo
- EA\_GenerateFromSchema

## EA\_GenerateFromSchema

Respond to a 'Generate' request from the **Schema Composer** when using the profile type specified by the **EA\_IsSchemaExporter** event. The **SchemaComposer** object can be used to traverse the schema. Export formats that have been requested by the user for generation will be listed in the **exports** parameter.

### Syntax

Sub **EA\_GenerateFromSchema** (Repository as EA.Repository, composer as EA.SchemaComposer, exports as String)

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
composer	Type: EA.SchemaComposer Direction: IN Description: Provides access to the types defined in the schema currently being generated. Use the <i>SchemaTypes</i> attribute to enumerate through the types and output to the appropriate export format.
exports	Type: String Direction: IN Description: Comma-separated list of export formats that the user has requested in the 'Generate' dialog.

### Return Value

None.

## EA\_GetProfileInfo

Add-ins can optionally implement this function to define the capabilities of the **Schema Composer** when working with the profile type specified by the EA\_IsSchemaExporter event.

### Syntax

Sub EA\_GetProfileInfo (Repository as EA.Repository, profile as EA.SchemaProfile)

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
profile	Type: EA.SchemaProfile Direction: IN Description: An EA.SchemaProfile object representing the currently active profile type. Call the <i>SetCapability</i> function to enable or disable various capabilities of the <b>Schema Composer</b> . Call the <i>AddExportFormat</i> function to define additional export formats that this profile will support.

### Return Value

None.

## EA\_IsSchemaExporter

Enterprise Architect **Add-Ins** can integrate with the **Schema Composer** by providing alternatives to offer users for the generation of schemas and sub models.

The Add-in must implement this function to be listed in the Schema Composer.

### Syntax

Function EA\_IsSchemaExporter(Repository as EA.Repository, ByRef displayName as String) As Boolean

Parameter	Details
Repository	Type: EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.
displayName	Type: String Direction: OUT Description: The name of the custom schema set that will be provided by this <b>Add-In</b> .

### Return Value

Return **True** to indicate that this **Add-In** will provide schema export functionality and be listed as a Schema Set when defining a new profile in the **Schema Composer**.

# Add-In License Management Events

Enterprise Architect **Add-Ins** can respond to events associated with **Add-In** License Management.

## License Management Events

Event
EA_AddinLicenseValidate
EA_AddinLicenseGetDescription
EA_GetSharedAddinName

## EA\_AddinLicenseValidate

When a user directly enters into the 'License Management' dialog a license key that doesn't match a Sparx Systems key, EA\_AddinLicenseValidate is broadcast to all Enterprise Architect **Add-Ins**, providing them with a chance to use the **Add-In** key to determine the level of functionality to provide. When a key is retrieved from the Sparx Systems **Keystore** only the target Add-In will be called with the key.

For the Add-In to validate itself against this key, the Add-In's EA\_AddinLicenseValidate handler should return confirmation that the license has been validated. As the EA\_AddinLicenseValidate event is broadcast to all Add-Ins, one license can validate many Add-Ins.

If an Add-In elects to handle a license key by returning a confirmation to EA\_AddinLicenseValidate, it is called upon to provide a description of the license key through the EA\_AddinLicenseGetDescription event. If more than one Add-In elects to handle a license key, the first Add-In that returns a confirmation to EA\_AddinLicenseValidate is queried for the license key description.

### Syntax

Function EA\_AddinLicenseValidate (Repository As EA.Repository, AddinKey As String) As Boolean

Parameter	Type
AddinKey	String Direction: IN Description: The Add-in license key that has been entered in the 'License Management' dialog.
Repository	EA.Repository Direction: IN Description: 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

Returns **True** if the license key is validated for the current **Add-In**. Returns **False** otherwise.

## EA\_AddinLicenseGetDescription

Before the Enterprise Architect 'License Management' dialog is displayed, EA\_AddInLicenseGetDescription is sent once for each **Add-In** key to the first Add-In that elected to handle that key.

The value returned by EA\_AddinLicenseGetDescription is used as the key's plain text description.

### Syntax

Function EA\_AddinLicenseGetDescription (Repository as EA.Repository, AddinKey as String) As String

Parameter	Type
AddinKey	String Direction: IN Description: The <b>Add-In</b> license key that Enterprise Architect requires a description for.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.

### Return Value

A String containing a plain text description of the provided AddinKey.

## EA\_GetSharedAddinName

As an **Add-In** writer you can distribute keys to your Add-In via the Enterprise Architect **Keystore**, provided that your keys are added using a prefix that allows the system to identify the Add-In to which they belong.

EA\_GetSharedAddinName is called to determine what prefix the Add-In is using. If a matching key is found in the keystore the 'License Management' dialog will display the name returned by EA\_AddinLicenseGetDescription to your users. Finally, when the user selects a key, that key will be passed to your Add-In to validate by calling EA\_AddinLicenseValidate.

### Syntax

Function EA\_GetSharedAddinName (Repository as EA.Repository) As String

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open model. Poll its members to retrieve model data and user interface status information.

### Return Value

A String containing a product name code for the provided **Add-In**, such as MYADDIN. This will be shown in plain text in any keys added to the keystore.

### Notes

Shared **Add-In** keys have the format:

EASK-YOURCODE-REALKEY

- EASK - Constant string that identifies a shared key for an Enterprise Architect Add-In
- YOURCODE - The code you select and verify with us:
  - Displayed to the administrator of the keystore
  - Recommended length of 6-10 characters
  - Contains ASCII characters 33-126, except for '-' (45)
- REALKEY - Encoding of the actual key or checksums
  - Recommended length of 8-32 characters
  - Contains ASCII characters 33-126

We recommend that you contact Sparx Systems directly with proposed values to ensure that you don't clash with any other **Add-Ins**.

For example, these keys would all be interpreted as belonging to an Add-In returning MYADDIN from this function:

- EASK-MYADDIN-Test
- EASK-MYADDIN-{7AC4D426-9083-4fa2-93B7-25E2B7FB8DC5}
- EASK-MYADDIN-7AC4D426-9083-4fa2-93B7
- EASK-MYADDIN-25E2B7FB8DC5
- EASK-MYADDIN-2hdFhKA5jf0GAjn92UvqAnxwC13dxQGJtH7zLHJ9Ym8=





## Compartment Events

Enterprise Architect **Add-Ins** can respond to various events associated with user-generated element compartments.

### Compartment Broadcast Events

Event
EA_QueryAvailableCompartments
EA_GetCompartmentData

## EA\_QueryAvailableCompartments

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 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 this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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

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
```



## EA\_GetCompartmentData

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 these parameters.

Parameter	Type
oType	ObjectType Direction: IN Description: The type of the element for which data is being requested.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
sCompartment	String Direction: IN Description: The name of the compartment for which data is being requested.
sGUID	String Direction: IN Description: The GUID of the element for which data is being requested.

### Return Value

A variant containing a formatted string. The format is illustrated in this example:

### 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

```



## Context Item Events

Enterprise Architect **Add-Ins** can respond to events associated with changing context.

### Context Item Broadcast Events

Event
EA_OnContextItemChanged
EA_OnContextItemDoubleClicked
EA_OnNotifyContextItemModified



## EA\_OnContextItemChanged

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 in the same way as EA\_FileOpen.

### Syntax

Sub EA\_OnContextItemChanged (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA\_OnContextItemChanged function syntax contains these parameters.

Parameter	Type
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the <code>ot</code> parameter: <ul style="list-style-type: none"><li>• <code>ot (ObjectType)</code> - GUID value</li><li>• <code>otElement</code> - Element.ElementGUID</li><li>• <code>otPackage</code> - Package.PackageGUID</li><li>• <code>otDiagram</code> - Diagram.DiagramGUID</li><li>• <code>otAttribute</code> - Attribute.AttributeGUID</li><li>• <code>otMethod</code> - Method.MethodGUID</li><li>• <code>otConnector</code> - Connector.ConnectorGUID</li><li>• <code>otRepository</code> - NOT APPLICABLE, the GUID is an empty string</li></ul>
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnContextItemDoubleClicked

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 the **Enter key**) on the item in context, either in a diagram, in the **Project Browser** or in a custom compartment. Add-Ins to handle events can subscribe to this broadcast function.

### Syntax

Function EA\_OnContextItemDoubleClicked (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA\_OnContextItemDoubleClicked function syntax contains these parameters.

Parameter	Type
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the ot parameter: <ul style="list-style-type: none"><li>• otElement - Element.ElementGUID</li><li>• otPackage - Package.PackageGUID</li><li>• otDiagram - Diagram.DiagramGUID</li><li>• otAttribute - Attribute.AttributeGUID</li><li>• otMethod - Method.MethodGUID</li><li>• otConnector - Connector.ConnectorGUID</li></ul>
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.
Repository	EA.Repository Direction: IN Description: 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

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.

## EA\_OnNotifyContextItemModified

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.

### Syntax

Sub EA\_OnNotifyContextItemModified (Repository As EA.Repository, GUID As String, ot as EA.ObjectType)

The EA\_OnNotifyContextItemModified function syntax contains these parameters.

Parameter	Type
GUID	String Direction: IN Description: Contains the GUID of the new context item. The value corresponds to these properties, depending on the value of the ot parameter: <ul style="list-style-type: none"><li>• ot(ObjectType) - GUID value</li><li>• otElement - Element.ElementGUID</li><li>• otPackage - Package.PackageGUID</li><li>• otDiagram - Diagram.DiagramGUID</li><li>• otAttribute - Attribute.AttributeGUID</li><li>• otMethod - Method.MethodGUID</li><li>• otConnector - Connector.ConnectorGUID</li></ul>
ot	EA.ObjectType Direction: IN Description: Specifies the type of the new context item.
Repository	EA.Repository Direction: IN Description: 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.

## EA\_FileClose

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).

### Syntax

Sub EA\_FileClose (Repository As EA.Repository)

The EA\_FileClose function syntax contains this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository 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.

## EA\_FileNew

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).

### Syntax

Sub EA\_FileNew (Repository As EA.Repository)

The EA\_FileNew function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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.

## EA\_FileOpen

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).

### Syntax

Sub EA\_FileOpen (Repository As EA.Repository)

The EA\_FileOpen function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnPostCloseDiagram

EA\_OnPostCloseDiagram notifies **Add-Ins** that a diagram has been closed.

### Syntax

Function EA\_OnPostCloseDiagram (Repository As EA.Repository, DiagramID As Integer)

The EA\_OnPostCloseDiagram function syntax contains these parameters.

Parameter	Type
DiagramID	Integer Direction: IN Description: Contains the Diagram ID of the diagram that was closed.
Repository	EA.Repository Direction: IN Description: An EA.Repository 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.

## EA\_OnPostInitialized

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.

### Syntax

Sub EA\_OnPostInitialized (Repository As EA.Repository)

The EA\_OnPostInitialized function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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.



## EA\_OnPostOpenDiagram

EA\_OnPostOpenDiagram notifies **Add-Ins** that a diagram has been opened.

### Syntax

Function EA\_OnPostOpenDiagram (Repository As EA.Repository, DiagramID As Integer)

The EA\_OnPostOpenDiagram function syntax contains these parameters.

Parameter	Type
DiagramID	Integer Direction: IN Description: Contains the Diagram ID of the diagram that was opened.
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnPostTransform

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty Objects for the transform performed: <ul style="list-style-type: none"><li>• Transform: A string value corresponding to the name of the transform used</li><li>• PackageID: A long value corresponding to Package.PackageID of the destination Package</li></ul>
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnPreExitInstance

EA\_OnPreExitInstance is not currently used.

### Syntax

Sub EA\_OnPreExitInstance (Repository As EA.Repository)

The EA\_OnPreExitInstance function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnRetrieveModelTemplate

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.

### Syntax

Function EA\_OnRetrieveModelTemplate (Repository As EA.Repository, sLocation As String) As String

The EA\_OnRetrieveModelTemplate function syntax contains these parameters.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
sLocation	String Direction: IN Description: The name of the template requested; this should match the location attribute in the <ModelTemplates> section of an MDG Technology File.

### Return Value

Return a string containing the XMI export of the model that is being used as a template. Return an empty string if access to the template is denied; the **Add-In** is to handle user notification of the error.

### 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
```

## EA\_OnTabChanged

EA\_OnTabChanged notifies **Add-Ins** that the currently open tab has changed.

Diagrams do not generate the message when they are first opened - use the broadcast event EA\_OnPostOpenDiagram for this purpose.

### Syntax

Function EA\_OnTabChanged (Repository As EA.Repository, TabName As String, DiagramID As Integer)

The EA\_OnTabChanges function syntax contains these parameters.

Parameter	Type
DiagramID	Long Direction: IN Description: The diagram ID, or 0 if switched to an <b>Add-In</b> tab.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TabName	String Direction: IN Description: The name of the tab to which focus has been switched.

### Return Value

None

# 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 these broadcast events.

Command	Detail
EA_OnInitializeUserRules	EA_OnInitializeUserRules is intercepted in order to define rule categories and rules.
EA_OnStartValidation	EA_OnStartValidation can be intercepted to perform any required processing prior to validation.
EA_OnEndValidation	EA_OnEndValidation can be intercepted to perform any required clean-up after validation has completed.
Validate Request	These functions intercept each request to validate an individual element, Package, diagram, connector, attribute and method.
Validate Element	EA_OnRunElementRule
Validate Package	EA_OnRunPackageRule
Validate Diagram	EA_OnRunDiagramRule
Validate Connector	EA_OnRunConnectorRule
Validate Attribute	EA_OnRunAttributeRule
Validate Method	EA_OnRunMethodRule

## EA\_OnInitializeUserRules

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* topic.

### Syntax

Sub EA\_OnInitializeUserRules (Repository As EA.Repository)

The EA\_OnInitializeUserRules function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.



## EA\_OnStartValidation

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 these parameters.

Parameter	Type
Args	ParamArray of Variant Direction: IN Description: Contains a list of Rule Categories that are active for the current invocation of model validation.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## EA\_OnEndValidation

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 these parameters.

Parameter	Type
Args	ParamArray of Variant Direction: IN Description: Contains a list of Rule Categories that were active for the invocation of model validation that has just completed.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## EA\_OnRunElementRule

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.

### Syntax

Sub EA\_OnRunElementRule (Repository As EA.Repository, RuleID As String, Element As EA.Element)

The EA\_OnRunElementRule function syntax contains these parameters.

Parameter	Type
Element	EA.Element Direction: IN Description: The element to potentially perform validation on.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.

## EA\_OnRunPackageRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
PackageID	Long Direction: IN Description: The ID of the Package to potentially perform validation on. Use the 'Repository.GetPackageByID' method to retrieve the Package object.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' method.

## EA\_OnRunDiagramRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
DiagramID	Long Direction: IN Description: The ID of the diagram to potentially perform validation on. Use the Repository.GetDiagramByID method to retrieve the diagram object.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.

## EA\_OnRunConnectorRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
ConnectorID	Long Direction: IN Description: The ID of the connector to potentially perform validation on. Use the 'Repository.GetConnectorByID' method to retrieve the connector object.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.

## EA\_OnRunAttributeRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
AttributeGUID	String Direction: IN Description: The GUID of the attribute to potentially perform validation on. Use the 'Repository.GetAttributeByGuid' method to retrieve the attribute object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given attribute. Use the 'Repository.GetElementByID' method to retrieve the object.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.

## EA\_OnRunMethodRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
MethodGUID	String Direction: IN Description: The GUID of the method to potentially perform validation on. Use the 'Repository.GetMethodByGuid' method to retrieve the method object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given method. Use the 'Repository.GetElementByID' method to retrieve the object.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.



## EA\_OnRunParameterRule

This event is triggered once for each rule defined in EA\_OnInitializeUserRules 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 these parameters.

Parameter	Type
MethodGUID	String Direction: IN Description: The GUID of the method that owns the given parameter. Use the 'Repository.GetMethodByGuid' method to retrieve the method object.
ObjectID	Long Direction: IN Description: The ID of the object that owns the given parameter. Use the 'Repository.GetElementByID' method to retrieve the object.
ParameterGUID	String Direction: IN Description: The GUID of the parameter to potentially perform validation on. Use this to retrieve the parameter by iterating through the 'Method.Parameters' collection.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String Direction: IN Description: The ID that was passed into the 'Project.DefineRule' command.

# Model Validation Example

This 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 lConnectorID)
{
    EA.Connector Connector = Repository.GetConnectorByID((int)lConnectorID);
    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 lDiagramID)
{
    EA.Diagram Diagram = Repository.GetDiagramByID((int)lDiagramID);
    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));
}
}
}
```

## Post-New Events

Enterprise Architect **Add-Ins** can respond to the creation of new elements, connectors, objects, attributes, methods and Packages using these broadcast events:

### Post-New Broadcast Events

Event
EA_OnPostNewElement
EA_OnPostNewConnector
EA_OnPostNewDiagram
EA_OnPostNewDiagramObject
EA_OnPostNewAttribute
EA_OnPostNewMethod
EA_OnPostNewPackage
EA_OnPostNewGlossaryTerm



## EA\_OnPostNewElement

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 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 'Properties' dialog.

### Syntax

Function EA\_OnPostNewElement (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewElement function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new element: <ul style="list-style-type: none"><li>ElementID: A long value corresponding to Element.ElementID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the element has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewConnector

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 Toolbox or Resources window onto a diagram. The notification is provided immediately after the connector is added to the model.

### Syntax

Function EA\_OnPostNewConnector (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewConnector function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new connector: <ul style="list-style-type: none"><li>ConnectorID: A long value corresponding to Connector.ConnectorID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the connector has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewDiagram

EA\_OnPostNewDiagram notifies **Add-Ins** that a new diagram has been created. It enables Add-Ins to modify the diagram upon creation.

### Syntax

Function EA\_OnPostNewDiagram (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewDiagram function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new diagram: <ul style="list-style-type: none"><li>DiagramID: A long value corresponding to Diagram.PackageID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the diagram has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewDiagramObject

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 **Project Browser** or Resources window onto a diagram. The notification is provided immediately after the object is added to the diagram.

### Syntax

Function EA\_OnPostNewDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewDiagramObject function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the new element: <ul style="list-style-type: none"><li>• ID: A long value corresponding to the ElementID of the object that has been added to the diagram</li><li>• DiagramID: A long value corresponding to the DiagramID of the diagram to which the object has been added</li><li>• DUID: A string value for the DUID; can be used with Diagram.GetDiagramObjectByID to retrieve the new DiagramObject</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the element has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewAttribute

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-and-dropping from the **Project Browser**, using the 'Attributes' tab of the 'Features' dialog, or using the in-place editor on the diagram. The notification is provided immediately after the attribute is created.

### Syntax

Function EA\_OnPostNewAttribute (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewAttribute function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new attribute: <ul style="list-style-type: none"><li>AttributeID: A long value corresponding to Attribute.AttributeID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the attribute has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewMethod

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.

### Syntax

Function EA\_OnPostNewMethod (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewMethod function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new method: <ul style="list-style-type: none"><li>MethodID: A long value corresponding to Method.MethodID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the method has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewPackage

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 Toolbox or Resources window onto a diagram, or by selecting the **New Package icon** from the **Project Browser**.

### Syntax

Function EA\_OnPostNewPackage (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewPackage function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the new Package: <ul style="list-style-type: none"><li>PackageID: A long value corresponding to Package.PackageID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the Package has been updated during this notification. Return **False** otherwise.

## EA\_OnPostNewGlossaryTerm

EA\_OnPostNewGlossaryTerm notifies **Add-Ins** that a new glossary term has been created. It enables Add-Ins to modify the glossary term upon creation.

The notification is provided immediately after the glossary term is added to the model.

### Syntax

Function EA\_OnPostNewGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPostNewGlossaryTerm function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the new glossary term: <ul style="list-style-type: none"><li>• TermID: A string value corresponding to Term.TermID</li><li>• Term: A string value corresponding to the name of the glossary term being created</li><li>• Meaning: A string value corresponding to meaning of the glossary term being created</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the glossary term has been updated during this notification. Return **False** otherwise.



## Pre-Deletion Events

Enterprise Architect **Add-Ins** can respond to requests to delete elements, attributes, methods, connectors, diagrams, Packages and glossary terms using these broadcast events:

### Pre-Deletion Broadcast Events

Event
EA_OnPreDeleteElement
EA_OnPreDeleteAttribute
EA_OnPreDeleteMethod
EA_OnPreDeleteConnector
EA_OnPreDeleteDiagram
EA_OnPreDeletePackage
EA_OnPreDeleteGlossaryTerm
EA_OnPreDeleteTechnology (Deprecated)

## EA\_OnPreDeleteElement

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the element to be deleted: <ul style="list-style-type: none"><li>• ElementID: A long value corresponding to Element.ElementID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the element from the model. Return **False** to disable deletion of the element.

## EA\_OnPreDeleteAttribute

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the attribute to be deleted: <ul style="list-style-type: none"><li>AttributeID: A long value corresponding to Attribute.AttributeID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the attribute from the model. Return **False** to disable deletion of the attribute.

## EA\_OnPreDeleteMethod

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the method to be deleted: <ul style="list-style-type: none"><li>MethodID: A long value corresponding to Method.MethodID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the method from the model. Return **False** to disable deletion of the method.

## EA\_OnPreDeleteConnector

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the connector to be deleted: <ul style="list-style-type: none"><li>ConnectorID: A long value corresponding to Connector.ConnectorID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the connector from the model. Return **False** to disable deletion of the connector.

## EA\_OnPreDeleteDiagram

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the diagram to be deleted: <ul style="list-style-type: none"><li>DiagramID: A long value corresponding to Diagram.DiagramID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the diagram from the model. Return **False** to disable deletion of the diagram.

## EA\_OnPreDeleteDiagramObject

EA\_OnPreDeleteDiagramObject notifies **Add-Ins** that a diagram object 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 attempts to permanently delete an element from 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\_OnPreDeleteDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreDeleteDiagramObject function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the element to be deleted: <ul style="list-style-type: none"><li>ID: A long value corresponding to DiagramObject.ElementID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the element from the model. Return **False** to disable deletion of the element.

## EA\_OnPreDeletePackage

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 these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the Package to be deleted: <ul style="list-style-type: none"><li>PackageID: A long value corresponding to Package.PackageID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the Package from the model. Return **False** to disable deletion of the Package.



## EA\_OnPreDeleteGlossaryTerm

EA\_OnPreDeleteGlossaryTerm notifies **Add-Ins** that a glossary term is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the glossary term.

The notification is provided immediately before the glossary term is deleted, so that the **Add-In** can disable deletion of the glossary term.

### Syntax

Function EA\_OnPreDeleteGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreDeleteGlossaryTerm function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the glossary term to be deleted: <ul style="list-style-type: none"><li>TermID: A long value corresponding to Term.TermID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the glossary term from the model. Return **False** to disable deletion of the glossary term.

## Pre New-Object Events

When you create an **Add-In**, you can include broadcast events to intercept and respond to requests to create new objects, including elements, connectors, diagram objects, attributes, methods and Packages.

### Events to intercept

Event
Creation of a new element
Creation of a new connector
Creation of a new diagram
Creation of a new diagram object
Creation of a new element by dropping onto a diagram from the <b>Project Browser</b> .
Creation of a new attribute
Creation of a new method
Creation of a new Package
Creation of a new glossary term

## EA\_OnPreNewElement

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 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.

### Syntax

Function EA\_OnPreNewElement (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewElement function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the element to be created: <ul style="list-style-type: none"><li>• Type: A string value corresponding to Element.Type</li><li>• FQStereotype: A string value corresponding to Element.FQStereotype</li><li>• Stereotype: A string value corresponding to Element.Stereotype</li><li>• ParentID: A long value corresponding to Element.ParentID</li><li>• DiagramID: A long value corresponding to the ID of the diagram to which the element is being added</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new element to the model. Return **False** to disable addition of the new element.

## EA\_OnPreNewConnector

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 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.

### Syntax

Function EA\_OnPreNewConnector (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewConnector function syntax contains these elements:

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the connector to be created: <ul style="list-style-type: none"><li>• Type: A string value corresponding to Connector.Type</li><li>• Subtype: A string value corresponding to Connector.Subtype</li><li>• Stereotype: A string value corresponding to Connector.Stereotype</li><li>• ClientID: A long value corresponding to Connector.ClientID</li><li>• SupplierID: A long value corresponding to Connector.SupplierID</li><li>• DiagramID: A long value corresponding to Connector.DiagramID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new connector to the model. Return **False** to disable addition of the new connector.

## EA\_OnPreNewDiagram

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.

### Syntax

Function EA\_OnPreNewDiagram (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewDiagram function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the diagram to be created: <ul style="list-style-type: none"><li>• Type: A string value corresponding to Diagram.Type</li><li>• ParentID: A long value corresponding to Diagram.ParentID</li><li>• PackageID: A long value corresponding to Diagram.PackageID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new diagram to the model. Return **False** to disable addition of the new diagram.

## EA\_OnPreNewDiagramObject

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.

### Syntax

Function EA\_OnPreNewDiagramObject (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewDiagramObject function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the object to be created: <ul style="list-style-type: none"><li>• Type: A string value corresponding to the Type of object being added to the diagram</li><li>• Stereotype: A string value corresponding to the Stereotype of the object being added to the diagram</li><li>• ID: A long value corresponding to the ID of the Element, Package or Diagram being added to the diagram</li><li>• DiagramID: A long value corresponding to the ID of the diagram to which the object is being added</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the object to the model. Return **False** to disable addition of the object.

## EA\_OnPreDropFromTree

When a user drags any kind of element from the **Project Browser** onto a diagram, EA\_OnPreDropFromTree notifies the **Add-In** that a new item is about to be dropped onto a diagram. The notification is provided immediately before the element is dropped, so that the Add-In can override the default action that would be taken for this drag.

### Syntax

Function EA\_OnPreDropFromTree (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreDropFromTree function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the element to be created: <ul style="list-style-type: none"><li>• ID: A long value of the type being dropped</li><li>• Type: A string value corresponding to type of element being dropped</li><li>• DiagramID: A long value corresponding to the ID of the diagram to which the element is being added</li><li>• PositionX: The X coordinate into which the element is being dropped</li><li>• PositionY: The Y coordinate into which the element is being dropped</li><li>• DroppedID: A long value corresponding to the ID of the element the item has been dropped onto</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to allow the default behavior to be executed. Return **False** if you are overriding this behavior.

## EA\_OnPreNewAttribute

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' tab of the 'Features' 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.

### Syntax

Function EA\_OnPreNewAttribute (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewAttribute function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the attribute to be created: <ul style="list-style-type: none"><li>• Type: A string value corresponding to Attribute.Type</li><li>• Stereotype: A string value corresponding to Attribute.Stereotype</li><li>• ParentID: A long value corresponding to Attribute.ParentID</li><li>• ClassifierID: A long value corresponding to Attribute.ClassifierID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new attribute to the model. Return **False** to disable addition of the new attribute.



## EA\_OnPreNewMethod

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 'Operations' tab of the 'Features' 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.

### Syntax

Function EA\_OnPreNewMethod (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewMethod function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the method to be created: <ul style="list-style-type: none"><li>• ReturnType: A string value corresponding to Method.ReturnType</li><li>• Stereotype: A string value corresponding to Method.Stereotype</li><li>• ParentID: A long value corresponding to Method.ParentID</li><li>• ClassifierID: A long value corresponding to Method.ClassifierID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new method to the model. Return **False** to disable addition of the new method.

## EA\_OnPreNewPackage

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 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.

### Syntax

Function EA\_OnPreNewPackage (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewPackage function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the Package to be created: <ul style="list-style-type: none"><li>• Stereotype: A string value corresponding to Package.Stereotype</li><li>• ParentID: A long value corresponding to Package.ParentID</li><li>• DiagramID: A long value corresponding to the ID of the diagram to which the Package is being added</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new Package to the model. Return **False** to disable addition of the new Package.

## EA\_OnPreNewGlossaryTerm

EA\_OnPreNewGlossaryTerm notifies **Add-Ins** that a new glossary term is about to be created. It enables Add-Ins to permit or deny creation of the new glossary term.

The notification is provided immediately before the glossary term is created, so that the **Add-In** can disable addition of the element.

### Syntax

Function EA\_OnPreNewGlossaryTerm (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreNewGlossaryTerm function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the glossary term to be created: <ul style="list-style-type: none"><li>• TermID: A string value corresponding to Term.TermID</li><li>• Term: A string value corresponding to the name of the glossary term being created</li><li>• Meaning: A string value corresponding to meaning of the glossary term being created</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable addition of the new glossary term to the model. Return **False** to disable addition of the new glossary term.


## Tagged Value Broadcasts

Enterprise Architect includes the Addin Broadcast **Tagged Value** type that allows an **Add-In** to respond to attempts to edit it. The function that is called depends on the type of object the Tagged Value is on.

### Tagged Value Broadcast Events

Event
EA_OnAttributeTagEdit
EA_OnConnectorTagEdit
EA_OnElementTagEdit
EA_OnMethodTagEdit

## EA\_OnAttributeTagEdit

EA\_OnAttributeTagEdit is called when the user clicks the  button for a **Tagged Value** of type AddinBroadcast on an attribute.

The **Add-In** displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.


### Syntax

Sub EA\_OnAttributeTagEdit (Repository As EA.Repository, AttributeID As Long, String TagName, String TagValue, String TagNotes)

The EA\_OnAttributeTagEdit function syntax contains these parameters.

Parameter	Type
AttributeID	Long Direction: IN Description: The ID of the attribute that this <b>Tagged Value</b> is on.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TagName	String Direction: IN Description: The name of the <b>Tagged Value</b> to edit.
TagNotes	String Direction: INOUT Description: The current value of the <b>Tagged Value</b> notes; if the value is updated, the new value is stored in the repository on exit of the function.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.

## EA\_OnConnectorTagEdit

EA\_OnConnectorTagEdit is called when the user clicks the  button for a **Tagged Value** of type AddinBroadcast on a connector.

The **Add-In** displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

### Syntax

Sub EA\_OnConnectorTagEdit (Repository As EA.Repository, ConnectorID As Long, String TagName, String TagValue, String TagNotes)

The EA\_OnConnectorTagEdit function syntax contains these parameters.

Parameter	Type
ConnectorID	Long Direction: IN Description: The ID of the connector that this <b>Tagged Value</b> is on.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TagName	String Direction: IN Description: The name of the <b>Tagged Value</b> to edit.
TagNotes	String Direction: INOUT Description: The current value of the <b>Tagged Value</b> notes; if the value is updated, the new value is stored in the repository on exit of the function.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.

## EA\_OnElementTagEdit

EA\_OnElementTagEdit is called when the user clicks the  button for a **Tagged Value** of type AddinBroadcast on an element.

The **Add-In** displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.


### Syntax

Sub EA\_OnElementTagEdit (Repository As EA.Repository, ObjectID As Long, String TagName, String TagValue, String TagNotes)

The EA\_OnElementTagEdit function syntax contains these elements:

Parameter	Type
ObjectID	Long Direction: IN Description: The ID of the object (element) that this <b>Tagged Value</b> is on.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TagName	String Direction: IN Description: The name of the <b>Tagged Value</b> to edit.
TagNotes	String Direction: INOUT Description: The current value of the <b>Tagged Value</b> notes; if the value is updated, the new value is stored in the repository on exit of the function.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.

## EA\_OnMethodTagEdit

EA\_OnMethodTagEdit is called when the user clicks the  button for a **Tagged Value** of type AddinBroadcast on an operation.

The **Add-In** displays fields to show and change the value and notes; this function provides the initial values for the Tagged Value notes and value, and takes on any changes on exit of the function.

### Syntax

Sub EA\_OnMethodTagEdit (Repository As EA.Repository, MethodID As Long, String TagName, String TagValue, String TagNotes)

The EA\_OnMethodTagEdit function syntax contains these elements:

Parameter	Type
MethodID	Long Direction: IN Description: The ID of the method that this <b>Tagged Value</b> is on.
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
TagName	String Direction: IN Description: The name of the <b>Tagged Value</b> to edit.
TagNotes	String Direction: INOUT Description: The current value of the <b>Tagged Value</b> notes; if the value is updated, the new value is stored in the repository on exit of the function.
TagValue	String Direction: INOUT Description: The current value of the tag; if the value is updated, the new value is stored in the repository on exit of the function.



# Technology Events

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

## Technology Broadcast Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology
EA_OnPreDeleteTechnology (Deprecated)
EA_OnDeleteTechnology (Deprecated)
EA_OnImportTechnology (Deprecated)

## EA\_OnInitializeTechnologies

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 this parameter:

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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

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
```

## EA\_OnPreActivateTechnology

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 or by selecting the technology in the list box in the Default Tools toolbar).

The notification is provided immediately after the user attempts to activate the MDG Technology, so that the **Add-In** can permit or disable activation of the Technology.

### Syntax

Function EA\_OnPreActivateTechnology (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreActivateTechnology function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the MDG Technology to be activated: <ul style="list-style-type: none"><li>TechnologyID: A string value corresponding to the MDG Technology ID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable activation of the MDG Technology resource in the model. Return **False** to disable activation of the MDG Technology resource.

## EA\_OnPostActivateTechnology

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, or by selecting the technology in the list box in the Default Tools toolbar).

The notification is provided immediately after the user succeeds in activating the MDG Technology, so that the **Add-In** can update the Technology if necessary.

### Syntax

Function EA\_OnPostActivateTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA\_OnPostActivateTechnology function syntax contains these parameters:

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects for the MDG Technology to be activated: <ul style="list-style-type: none"><li>TechnologyID: A string value corresponding to the MDG Technology ID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** if the MDG Technology resource is updated during this notification. Return **False** otherwise.

## EA\_OnPreDeleteTechnology

**Deprecated** - refers to deleting a technology through the Resources window; this process is no longer recommended. See *Deploy An MDG Technology* for information on recommended methods for using technologies.

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.

### Related Broadcast Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

### Syntax

Function EA\_OnPreDeleteTechnology (Repository As EA.Repository, Info As EA.EventProperties) As Boolean

The EA\_OnPreDeleteTechnology function syntax contains these elements:

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains this EventProperty object for the MDG Technology to be deleted: <ul style="list-style-type: none"><li>TechnologyID: A string value corresponding to the MDG Technology ID</li></ul>
Repository	EA.Repository Direction: IN Description: 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

Return **True** to enable deletion of the MDG Technology resource from the model. Return **False** to disable deletion of the MDG Technology resource.

## EA\_OnDeleteTechnology

**Deprecated** - refers to deleting a technology through the Resources window; this process is no longer recommended. See *Deploy An MDG Technology* for information of recommended methods for using technologies.

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.

### Related Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

### Syntax

Sub EA\_OnDeleteTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA\_OnDeleteTechnology function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects: <ul style="list-style-type: none"><li>TechnologyID: A string value corresponding to the MDG Technology ID</li></ul>
Repository	EA.Repository Direction: IN Description: 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.

## EA\_OnImportTechnology

**Deprecated** - refers to importing a technology into the Resources window; this process is no longer recommended. See *Deploy An MDG Technology* for information of recommended methods for using technologies.

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.

### Related Events

Event
EA_OnInitializeTechnologies
EA_OnPreActivateTechnology
EA_OnPostActivateTechnology

### Syntax

Sub EA\_OnImportTechnology (Repository As EA.Repository, Info As EA.EventProperties)

The EA\_OnImportTechnology function syntax contains these parameters.

Parameter	Type
Info	EA.EventProperties Direction: IN Description: Contains these EventProperty objects: <ul style="list-style-type: none"><li>TechnologyID: A string value corresponding to the MDG Technology ID</li></ul>
Repository	EA.Repository Direction: IN Description: 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.

## Custom Views

Enterprise Architect enables custom windows to be inserted as a Diagram Tab within the **Diagram View** that appears at the center of the Enterprise Architect frame.

Creating a custom view helps you to easily display a custom interface within Enterprise Architect, alongside other diagrams and built-in views for quick and easy access.

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



## Create a Custom View

A custom view must be designed as an ActiveX Custom Control and inserted via the **Automation Interface**. ActiveX Custom Controls can be created using most well-known programming tools, including Microsoft Visual Studio. 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 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 the Add-In is defined in the same OCX that provides the custom view.

This is a C# code example:

```
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 = {"Show Custom View", "Show Button"};
            return ret;
        }
    }
    public void EA_MenuClick(EA.Repository Rep, string Location, string MenuName, string ItemName)
    {
        if(ItemName == "Show Custom View")
            m_MyControl = (UserControl1) Rep.AddTab("C# Demo","ContDemo.UserControl1");
        else if(ItemName == "Show Button")
            m_MyControl.ShowButton();
    }
}
```

## Add a Portal

Enterprise Architect provides a set of Portals, each of which is a collection of shortcuts and information on performing specific areas of work on a project. The portals help both new and experienced users quickly identify and set up the facilities they most often use in their assigned tasks.

You can add your own Portal to the system-installed set, to provide a convenient and concise call-up of one or more groups of facilities available in your **Add-In**.

### Example Code

```
public String EA_LoadWindowManager(EA.Repository Repository)
{
    return Resource1.WindowManager;
}
```

Where Resource1.WindowManager is a resource file with these contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<perspectives>
  <perspective name="Add-in">
    <category name="Add-in" type="commandlist" projectrequired="true">
      <item name="Hello World" command="CallAddin" addin="CS_AddinFramework" function="HelloWorld"/>
      <item name="Model Dump" command="CallScript" group="Local Scripts" script="JScript - Recursive Model Dump
Example"/>
    </category>
    <category name="Open Diagrams" type="currentdiagramlist" state = "open"/>
    <category name="Recent Diagrams" type="recentdiagramlist" state = "open"/>
    <category name="Other Windows" type="otherwindowlist" state = "open"/>
  </perspective>
</perspectives>
```

Note that the **Add-In** cannot specify the icon used.

## Custom Docked Window

Custom docked windows can be added into the Enterprise Architect user interface. Once added, they can be shown and docked in the same way as other built-in Enterprise Architect docked windows.

A custom docked window must be designed as an ActiveX Custom Control and inserted via the automation interface. ActiveX Custom Controls can be created using most well-known programming tools, including Microsoft Visual Studio. 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 using the AddWindow() method of the Repository object. While it is possible to call AddWindow() from any automation client, it is likely that you would call it from an **Add-In**, and that the Add-In is defined in the same OCX that provides the custom view.

To view custom docked windows that have been added, select 'Extensions | Add-In Windows'.

Custom docked windows can also be made visible by the automation client or Add-in using the ShowAddinWindow() method, or hidden by using the HideAddinWindow() method.

This is an example in C# code:

```
public class Addin
{
    UserControl1 m_MyControl;
    public void EA_Connect(EA.Repository Rep)
    {
        m_MyControl = (UserControl1) Rep.AddWindow
            ("C# Demo","ContDemo.UserControl1");
    }
    public object EA_GetMenuItems(EA.Repository
    Repository, string Location, string MenuName)
    {
        if( MenuName == "" )
            return "-&C# Control Demo";
        else
        {
            String() ret = {"Show Window", "Show Button"};
            return ret;
        }
    }
    public void EA_MenuClick(EA.Repository Rep, string Location,
    string MenuName, string ItemName)
    {
        if( ItemName == "Show Window" )
            Rep.ShowAddinWindow("C# Demo");
        else if( ItemName == "Show Button" )
            m_MyControl.ShowButton();
    }
}
```



## 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. Two examples of MDG Add-Ins are the MDG Link for Eclipse and MDG Link for Visual Studio, both integrated with the Enterprise Architect installer.

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 event. 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 workstation.

MDG Add-Ins identify themselves as such during EA\_Connect 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.

## 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.

### MDGAdd-In Events

Event
MDG_BuildProject
MDG_Connect
MDG_Disconnect
MDG_GetConnectedPackages
MDG_GetProperty
MDG_Merge
MDG_NewClass
MDG_PostGenerate
MDG_PostMerge
MDG_PreGenerate
MDG_PreMerge
MDG_PreReverse
MDG_RunExe
MDG_View

## MDG\_Build Project

**Add-Ins** can use MDG\_BuildProject to handle file changes caused by generation. This function is called in response to a user selecting the 'Extensions | Build Project' menu option.

Respond to this event by compiling the project source files into a running application.

### Syntax

Sub MDG\_BuildProject (Repository As EA.Repository, PackageGuid As String)

The MDG\_BuildProject function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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.

## MDG\_Connect

An **Add-In** uses MDG\_Connect to handle a user driven request to connect a model branch to an external application. The function is called when the user attempts to connect a particular Enterprise Architect Package to an as yet unspecified external project. The Add-In calls the event 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 is 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.

### Syntax

Function MDG\_Connect (Repository As EA.Repository, PackageID as Long, PackageGuid As String) As Long

The MDG\_Connect function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The unique ID identifying the project provided by the <b>Add-In</b> when a connection to a project branch of an Enterprise Architect model was first established.
PackageID	Long Direction: IN Description: The PackageID of the Enterprise Architect Package the user has requested to have connected to an external project.
Repository	EA.Repository Direction: IN Description: 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

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.



## MDG\_Disconnect

**Add-Ins** can use MDG\_Disconnect 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.

### Syntax

Function MDG\_Disconnect (Repository As EA.Repository, PackageGuid As String) As Long

The MDG\_Disconnect function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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

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.

## MDG\_GetConnectedPackages

**Add-Ins** can use MDG\_GetConnectedPackages to return a list of current connections 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.

### Syntax

Function MDG\_GetConnectedPackages (Repository As EA.Repository) As Variant

The MDG\_GetConnectedPackages function syntax contains this parameter.

Parameter	Type
Repository	EA.Repository Direction: IN Description: 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

Returns an array of GUID strings representing individual Enterprise Architect Packages.

## MDG\_GetProperty

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 these `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  
`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  

```
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 these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
PropertyName	String Direction: IN Description: The name of the property that is used by Enterprise Architect. See above for the possible values.
Repository	EA.Repository Direction: IN Description: 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

See above.

## MDG\_Merge

**Add-Ins** can use MDG\_Merge 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 make the **Add-In** interact with the user to perform a merge between the model branch and the connected project.

### 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 these parameters.

Parameter	Type
ExportFiles	Variant Direction: OUT Description: A string array containing the list of files for each model object chosen for export by the <b>Add-In</b> . Each entry in this array must have a corresponding entry in the ExportObjects parameter at the same array index, so ExportFiles(2) must contain the filename of the object by ExportObjects(2).
ExportObjects	Variant Direction: OUT Description: The string array containing the list of new model objects (in Object ID format) to be exported by Enterprise Architect to the code project.
IgnoreLocked	String Direction: OUT Description: A value indicating whether to ignore any files locked by the code project (that is, ' <b>True</b> ' or ' <b>False</b> ').
ImportFiles	Variant Direction: OUT Description: 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	String Direction: OUT Description: The string value containing the name of the code language supported by the code project connected to the model branch.
PackageGuid	String

	Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
SynchObjects	Variant Direction: OUT Description: A string array containing a list of objects (Object ID format) to be jointly synchronized between the model branch and the project. See <i>Object ID Format</i> for the format of the Object IDs.
SynchType	String Direction: OUT Description: The value determining the user-selected type of synchronization to take place. See <i>Synchronize Type</i> for a list of valid values.

## Object ID Format

Each of the Object IDs listed in the 'SynchObjects' string arrays should have this format:

`((@namespace)*(#class)*($attribute|%operation|:property))*`

## 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 synchronizations are to occur

## MDG\_NewClass

**Add-Ins** can use MDG\_NewClass 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.

### 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 these parameters.

Parameter	Type
CodeID	String Direction: IN Description: A string used to identify the code element before it is created.
Language	String Direction: OUT Description: A string used to identify the programming language for the new Class. The language must be supported by Enterprise Architect.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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

Returns a string containing the file path that should be assigned to the Class.

## MDG\_PostGenerate

**Add-Ins** can use MDG\_PostGenerate 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, so the Add-Ins can 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.

### 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 these parameters.

Parameter	Type
FileContents	String Direction: IN Description: A string containing the proposed contents of the file.
FilePath	String Direction: IN Description: The path of the file Enterprise Architect intends to overwrite.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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

The return value depends on the type of event that this function is responding to (see introduction). This function is required to handle two separate and distinct cases.



## MDG\_PostMerge

MDG\_PostMerge is called by Enterprise Architect after a merge process has been completed.

File save checking should not be performed with this function, but should be handled by MDG\_PreGenerate, MDG\_PostGenerate and MDG\_PreReverse.

### Syntax

Function MDG\_PostMerge (Repository As EA.Repository, PackageGuid As String) As Long

The MDG\_PostMerge function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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

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.

## MDG\_PreGenerate

**Add-Ins** can use MDG\_PreGenerate 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.

### Return Value

Return a zero value to abort generation. Any other value enables the generation to continue.

### Syntax

Function MDG\_PreGenerate (Repository As EA.Repository, PackageGuid As String) As Long

The MDG\_PreGenerate function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## MDG\_PreMerge

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.

File save checking should not be performed with this function, but should be handled by MDG\_PreGenerate, MDG\_PostGenerate and MDG\_PreReverse.

### Syntax

Function MDG\_PreMerge (Repository As EA.Repository, PackageGuid As String) As Long

The MDG\_PreMerge function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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

A return value of zero indicates that the merge process can not occur. If the value is not zero the merge process proceeds.

If this method is not implemented then it is assumed that a merge process is used.

## MDG\_PreReverse

**Add-Ins** can use MDG\_PreReverse to save file changes before they are 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.

### Syntax

Sub MDG\_PreReverse (Repository As EA.Repository, PackageGuid As String, FilePaths As Variant)

The MDG\_PreReverse function syntax contains these parameters.

Parameter	Type
FilePaths	String array Direction: IN Description: An array of filepaths pointed to the files that are to be reverse engineered.
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: 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.

## MDG\_RunExe

**Add-Ins** can use MDG\_RunExe to run the target application.

This function is called when the user selects the 'Extensions | Run Exe' menu option.

Respond to this event by launching the compiled application.

### Return Value

None.

### Syntax:

Sub MDG\_RunExe (Repository As EA.Repository, PackageGuid As String)

The MDG\_RunExe function syntax contains these parameters.

Parameter	Type
PackageGuid	String Direction: IN Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b> .
Repository	EA.Repository Direction: IN Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## MDG\_View

**Add-Ins** can use MDG\_View to display user specified code elements.

This function is called by Enterprise Architect when the user asks to view a particular code element. The **Add-In** can then 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 these parameters.

Parameter	Type
CodeID	<p>String</p> <p>Direction: IN</p> <p>Description: Identifies the code element in this format:</p> <p style="padding-left: 40px;">&lt;type&gt;ElementPart&lt;type&gt;ElementPart...</p> <p>where each element is proceeded with a token identifying its type:</p> <p style="padding-left: 40px;">@ - namespace</p> <p style="padding-left: 40px"># - Class</p> <p style="padding-left: 40px;">\$ - attribute</p> <p style="padding-left: 40px;">% - operation</p> <p>For example, if a user has selected the m_Name attribute of Class1 located in namespace Name1, the Class ID would be passed through in this format:</p> <p style="padding-left: 40px;">@Name1#Class1%m_Name</p>
PackageGuid	<p>String</p> <p>Direction: IN</p> <p>Description: The GUID identifying the Enterprise Architect Package sub-tree that is controlled by the <b>Add-In</b>.</p>
Repository	<p>EA.Repository</p> <p>Direction: IN</p> <p>Description: An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.</p>

### 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.

