Rational. System Architect

IBM

VBA Extensibility Guide

IBM Rational
System Architect
VBA Extensibility Guide
Release 11.3

Before using this information, read the "Notices" in the Appendix, on page 20-1.

This edition applies to IBM® Rational® System Architect®, version 11.3 and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

1

Introduction

Introduction

This chapter covers the fundamentals of working with Rational System Architect and Visual Basic for Applications (VBA). You will learn about using macros, the VBA Editor, and the Object Browser.

Automating Rational System Architect

Rational System Architect is 'VBA enabled' allowing the user to control the Rational System Architect environment programmatically, and to control other applications using OLE automation.

Microsoft VBA and its development environment are installed with Rational System Architect. The programming environment, debugging environment, language and help system are the same as found in other VBA enabled applications, including Microsoft Office products.

Using automation, you can integrate other applications with Rational System Architect in the two ways. You can use Rational System Architect:

- As an automation controller, and call an OLE automation object from within a Rational System Architect script.
- As an automation server, and call its OLE automation object from within another OLE-compliant application.

Macros within Rational System Architect can be used to:

- Enhance the functionality of Rational System Architect – automating the presentation and rules checking of diagrams.
- Create diagrams, symbols and definitions from information contained in other applications.

Capture events that take place in Rational System Architect and store them in a file or database.

Programming Rational System Architect with VBA

Rational System Architect stores the current status of its VBA project environment in the initialization file SA2001.INI, (found in either the Windows or WinNt folder). The section entitled **Macros** may contain the following:

```
[Macros]
PermanentAutoLoad1=SAAuto.mac
TemporaryAutoLoad1= c:\program files\ibm \system
  architect suite\system
  architect\sample.mac|vxRead|vxShared|vxTransacted|b
  ProjectActive
RunTemporaryAutoMacros=T
OpenReadOnly=T
```

The above excerpt indicates that the permanent SAAUTO.MAC project file is loaded when Rational System Architect starts. (It will be hidden and read-only). The temporary SAMPLE.MAC project file is currently active and is also loaded in read-only mode. If any of the temporary project files contain an AutoExec module, then the macros are automatically executed. The default for opening new projects will be Read Only.

Running a Macro

To run the macros directly from the **Macros** dialog:

1. From the **Tools** menu, select the **Macros** submenu.
2. Select the **Run Macro…** option.

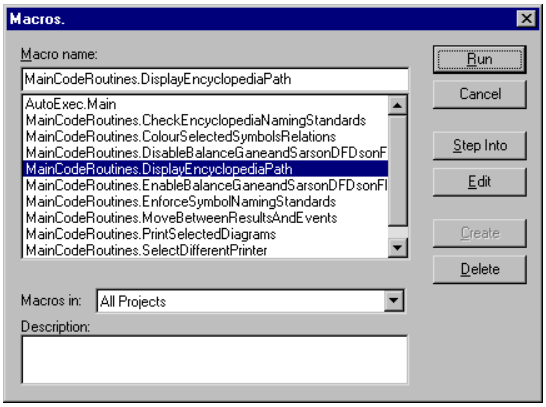


Figure 2-1. The Macros dialog

A macro can also be run from a Toolbar.

To add a macro to a toolbar:

- Right click over the MenuBar or any toolbar and select the **Customize…** option.
- Select the **Commands** tab and then the **Macros** option.
- Scroll through the available macros and **Drag and drop** the required macro onto a toolbar.

Macro Projects

Other Temporary Project files may be added to the Sample project.

To access the **Macro Projects** dialog:

1. Open the **Tools** menu
2. Select the **Macros** submenu
3. Select the **Macro Projects…** option

New project files may be added to the list of available projects or existing projects may be removed.

The **Enable automatic macros** checkbox can be deselected to disable the AutoExec macros.

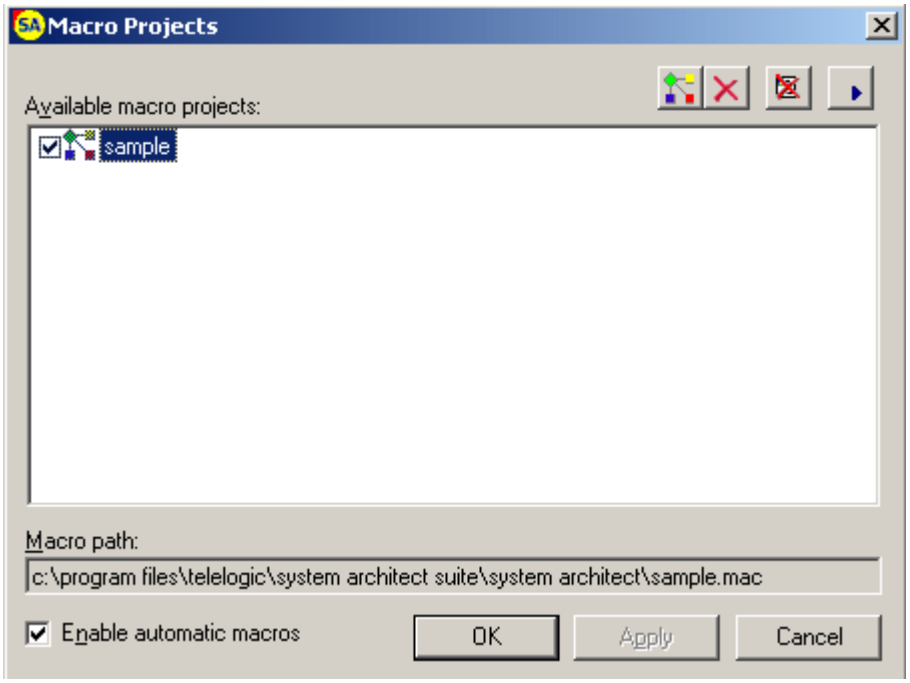


Figure 2-2. The Macro Projects dialog

Adding a New Project

Clicking on the **Add** command button will display the **Open Macro Project** dialog.

The Rational System Architect Project Files are recognized by their `.MAC` extension.

It is possible to select and open an existing Project or create a new Project by entering a File name that doesn't appear in the list.

If changes are to be made to the Project be sure to deselect the **Open as read-only** checkbox before opening the Project file.

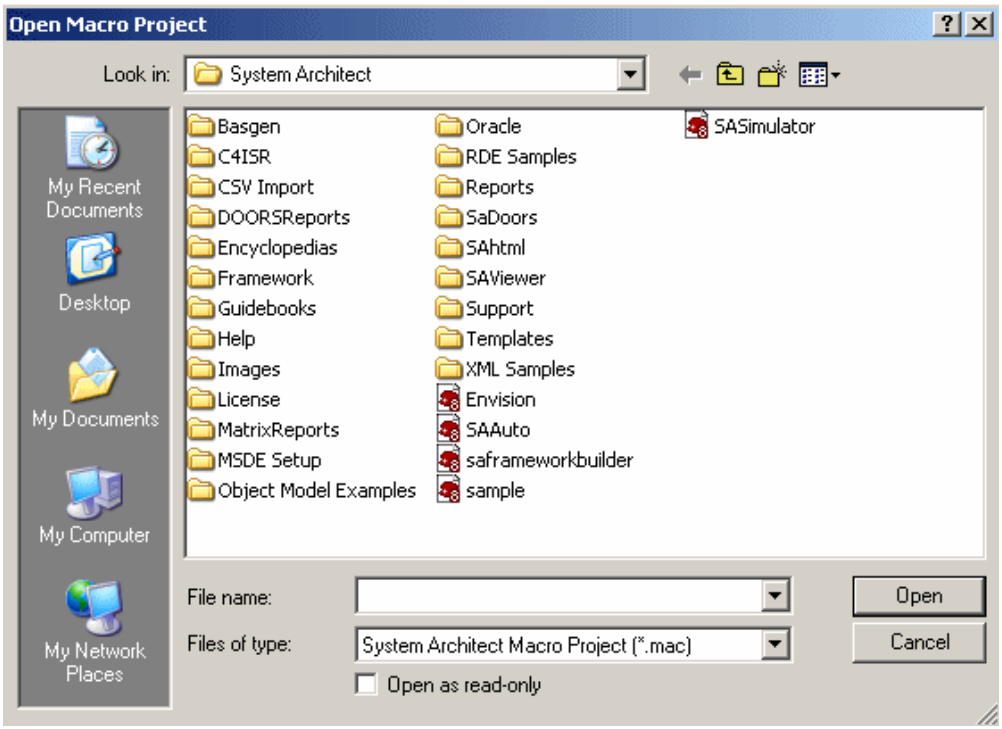


Figure 2-3. The Open Macro Projects dialog

Accessing the VBA Editor

In order to create a macro or modify an existing macro it is necessary to open the VBA Editor.

There are several ways to open up the VBA Editor:

- From the **Tools** menu, select the **Macros** submenu and then the VBA Editor option.
- Press Alt+F11.
- From the **Macros** dialog, click on the **Edit** command button.
- From the **Macro Projects** dialog, click on the **Apply command** button (if active) and then on the **Run Macro…** command button. This will open the **Macros** dialog where the **Edit** command button can be found.

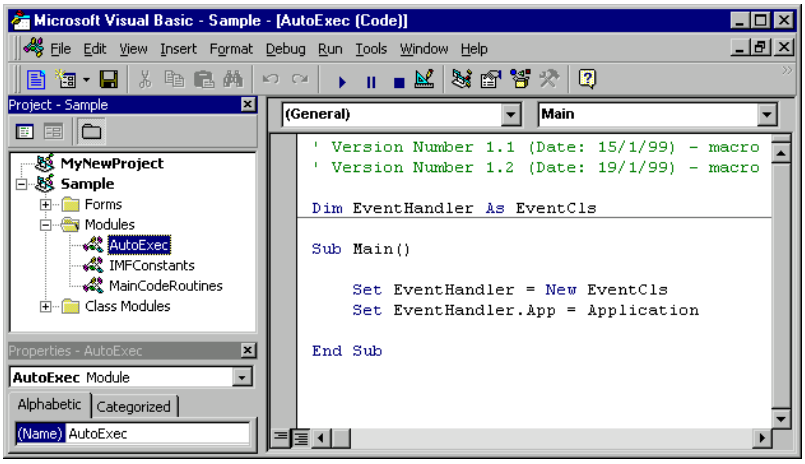


Figure 2-4. The VBA Editor

The Project Explorer

The VBA editor is made up of a number of windows. It is likely that the Project window will be visible when the VBA editor is first accessed.

To display the **Project** window:

1. From the **View** menu, select the **Project Explorer** option
2. The Sample project contains three **'Folder Groups'**, **Forms**, **Modules** and **Class Modules**.
3. Each folder may be opened to display its contents by clicking on the 'plus' symbol, or by double clicking on the folder or the name of the folder.

The contents of each folder will be displayed in alphabetical order, but the folders can be toggled off to reveal a complete alphabetical listing of all items.

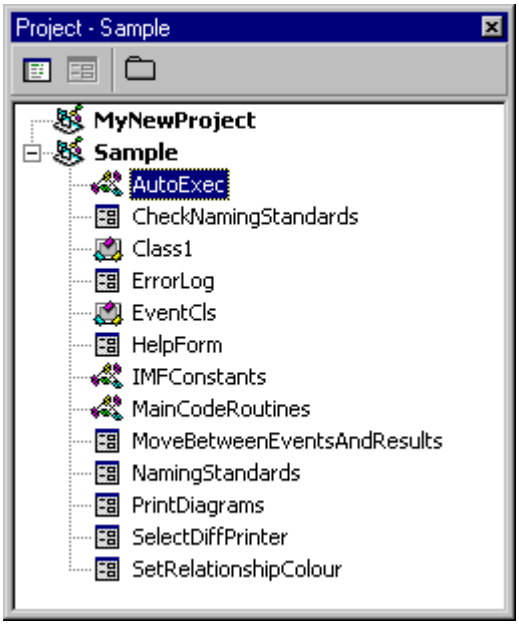


Figure 2-5. The Project Window

The Properties Window

If you click on any of the **Modules**, the **Properties** window will display that module's name. The name can then be changed within the **Properties** window.

If you double click on a **Module** name, its code will be displayed.

If you double click on the name of a **Form**, the **Form Object** will display, while the Properties window will display an alphabetical listing of the form's properties.

Right clicking on the name of a Form will give the option to **View** the **Form's Code**.

The **Properties** window can also display by category.

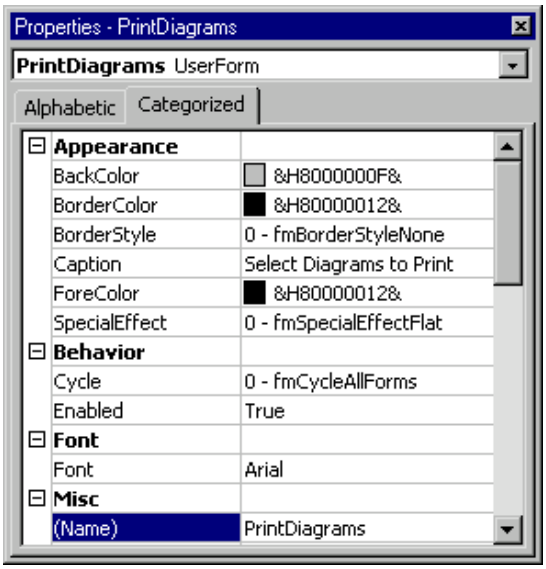


Figure 2-6. The Properties Window

Inserting Modules and Forms

Additional **Modules** and **Forms** may be added to any Project by using the **Insert** menu.

Selected **Modules** or **Forms** may be removed, and optionally **Exported**, by using the **Remove** option in the **File** menu.

Right clicking over the **Project** window will also give access to the **Insert** and **Remove** options.

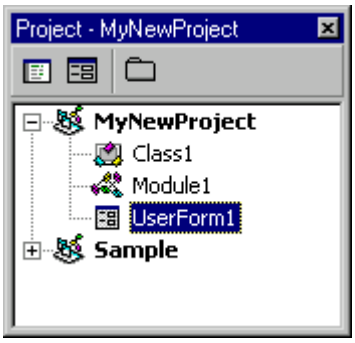


Figure 2-7. The Project Window

Object Browser

The Object Browser is a very useful feature of the VBA Editor, allowing the interrogation of available Object Libraries, Type Libraries and Dynamic Link Libraries.

Referencing Library Files

To make additional libraries available to the Object Browser:

- From the **Tools** menu, select the **References…** option.

It is expected that the SA2001 library will already be included in the list of selected references.

By scrolling down the list of Available References other references may be selected.

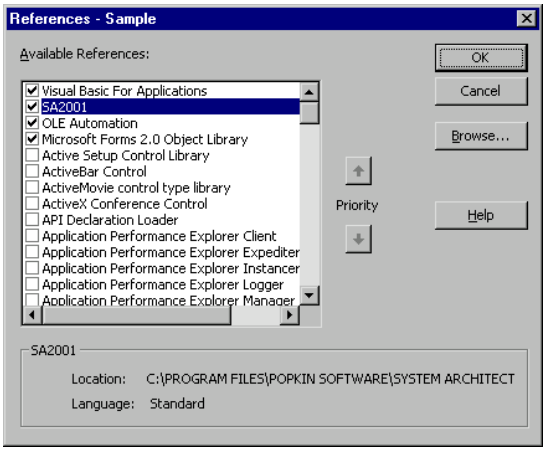


Figure 2-8. Selecting References

To display the Object Browser, either press the F2 function key or:

- From the **View** menu, select the **Object Browser** option

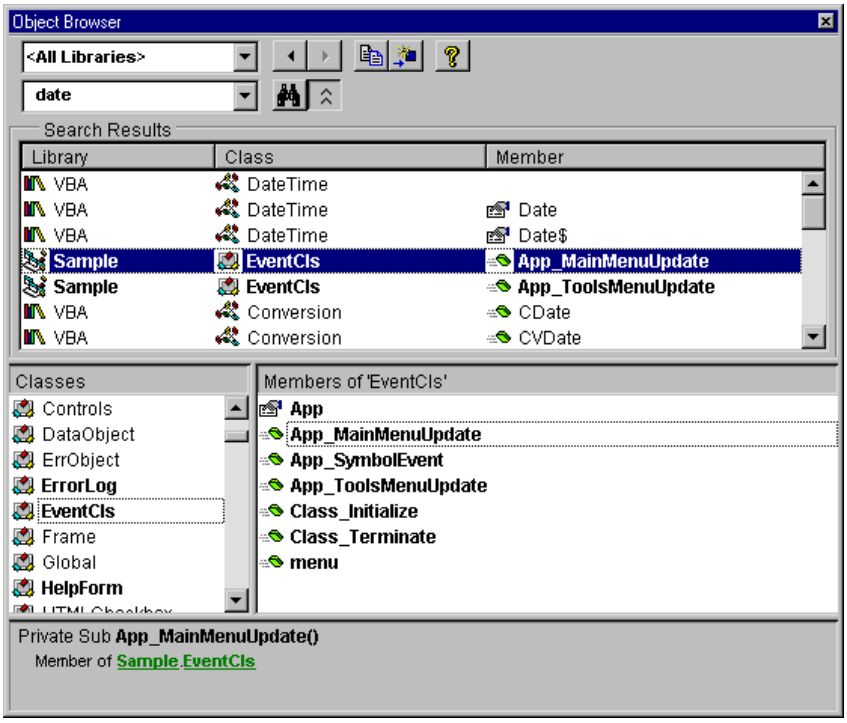


Figure 2-9. The Object Browser

The Object Browser may be filtered on specific libraries, and as shown above, used to search for items whose Class or Member name includes particular text.

If a project is selected in the Project Explorer window then the Object Browser will highlight all items that are part of that project.

Classes, Constants, Enumerated Types, Events, Global Variables, Methods, Modules, Properties, and User Defined Types will all be displayed, preceded by an appropriate icon.

2

Automation and Rational System Architect

Introduction

Automation is the capability of one application to declare and use objects that are actually created by different applications. By using Visual Basic for Applications (VBA), code may be written to access the objects of one or more of these application's objects in a single program.

The use of automation provides the potential for a fully integrated solution to be built based around the functionality of a number of products. Rational System Architect has VBA embedded within it, allowing other applications to use its functionality. The following sections of the manual explain this functionality.

This chapter will describe automation and how can it help provide a customizable solution. Then it will cover in greater detail the way Rational System Architect can be set up to provide the functionality required for the solution.

Automation

With automation, a programmer can incorporate objects from other applications into the current application to provide an integrated solution. For example, Rational System Architect might be used to create a Corporate Data Model built by a number of users but others wish to view the data with a specifically designed word report or excel spreadsheet. Using automation, an integrated word or excel report may be run directly from a custom menu item from within Rational System Architect. Automation allows the user to use tools built for specific tasks to be integrated.

Automation has also been called OLE Automation and sometimes ActiveX Automation in the past.

Automation Controller and Server

Automation is the one way to control another application's objects. Technically, one application holds the VBA code that controls the automation and the other provides the objects for use. The driver of this process is called the Automation Controller and the provider is called the Automation Server

Referencing the Type Library

VBA has a mechanism for understanding which object will be available within any VBA program. VBA will infer if you are creating the code in Rational System Architect that its library of automation objects is available, but it will not assume that any other type of libraries are also available. The user must then set a reference to the libraries that are required.

The type **library** is a database containing information about all the objects available for automation in any one application. This information would include details about the objects, attributes, events and methods available in the application. It is usually a separate file installed at the same time as the application, but in some cases may be supplied as part of the main executable file.

To reference this type library information, the user must be inside the VBA editor. This can be accessed from the relevant application menu or by simultaneously pressing the **Alt F11** keys. Once within the VBA editor, choose **Tools | References**.

In this example, the only referenced objects are those standard to VBA and Rational System Architect. To include other applications, scroll down and select the application of choice, for example Microsoft Excel 9.0 Object Library, which will then include all components of the Excel type library in the current application.

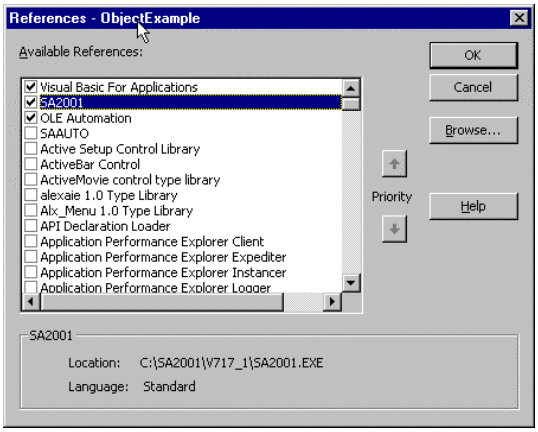


Figure 2-1. Choosing a Type Library

Viewing Automation Objects

Once a reference has been set to another application, then a list of the objects, attributes and methods of all the currently referenced type libraries may be viewed. The VBA editor has its own object browser to view all of these properties.

To access the VBA Object Browser, either press F2 or go to the **View | Object Browser** menu item from within the VBA Editor.

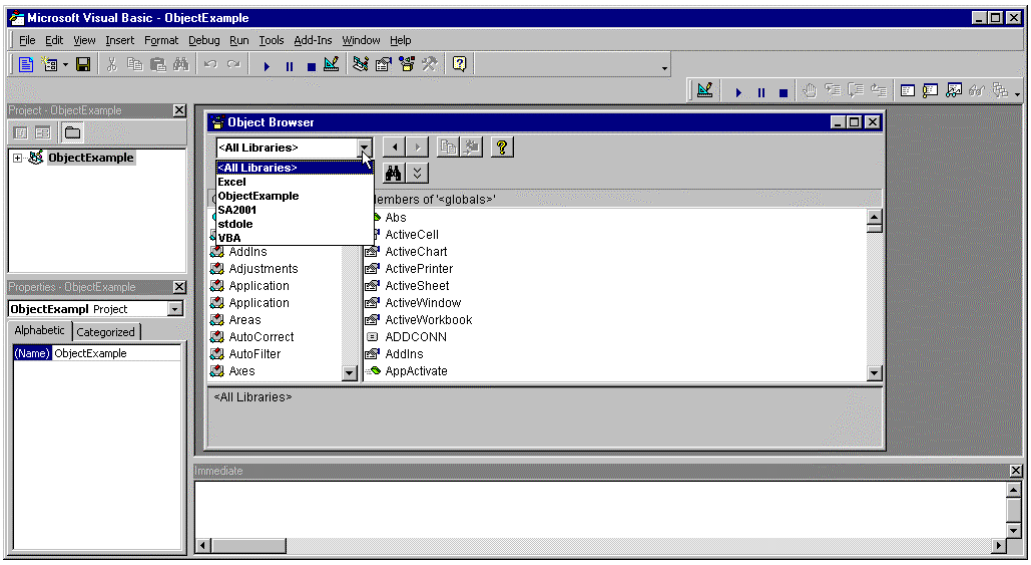


Figure 2-2. The VBA Object Browser

The object browser will list all libraries that have been selected in the type library references although an individual set of objects may be chosen. In the example, below we chose SA2001 as the type library we wish to look at and in particular the Definition class of automation objects.

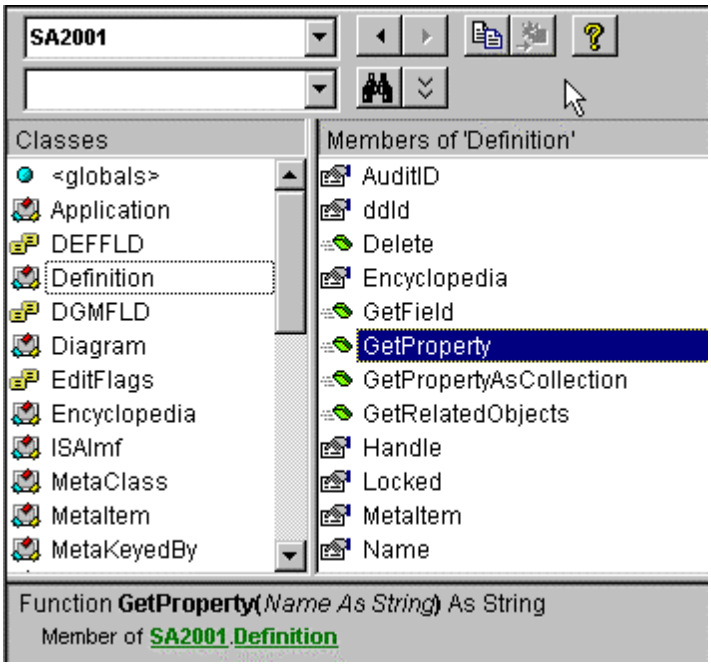


Figure 8-3. Browsing the SA2001 Type Library

The members of the Definition class are listed and **GetProperty** is selected. The parameters including type and return type are detailed in the lower browse window. By declaring a reference to an automation object all this internal application information becomes available to the VBA programmer for building an integrated solution.

Creating an Instance of the application

Although the required applications are referenced by VBA it is not until code is written that the application objects can be controlled. The next section looks at how to set up a new instance of an application to access its automated objects and how to declare it.

An instance is a session of the required application. In order to use automation objects the application must be resident in memory (but does not have to be visible). To access

objects, VBA uses the Dim and Set statements to declare the instance of an automation object just like a declaration would be made to a built in type. The only difference with automation is that at the point the objects are to be used, a new instance of the object must be created with the Set statement.

The following code declares the variable *ExObj* as an Excel Application. Note the full server.class name of the declaration – this ensures that the application class referred to is Excel's. Other applications may also have classes called application.

```
Dim ExObj as Excel.Application
```

The following line creates a new instance of the Excel Application and also sets the point in the code that the instantiation takes place.

```
Set ExObj = New Excel.Application
```

The variable *ExObj* can now access the Excel Automation Objects, for example displaying the standard **Open** dialog box from Excel using the automation object **GetOpenFilename()** filtered for text files with the following code.

```
fileToOpen = ExObj.GetOpenFilename("Text Files
  (*.txt), *.txt")
```

So all of Excel's Automation Objects are available alongside all of Rational System Architect's.

Releasing the Application Instance

Automation object instances are created with set, manipulated with automation function calls and then terminated. Terminating the automation class ensures that resources are freed from memory.

The following code should be used when the automation object is no longer used. Effectively closing the automation server.

```
Set Exobj = Nothing
```

In summary

Dim Object = Server.class	Declare Server
Set Object = New Server.class	Instantiation of Server
Automation Code	VBA code that accesses Server automation objects
Dim Object = Nothing	Release Automation server from Memory

Customizable Solutions

The previous section introduced the concepts of automation, describing how objects from some applications can be integrated with others. Any product that supports VBA and automation could be thought of as a customizable solution. It means that any product bought off the shelf that supports VBA may be modified as desired. This does not, however, refer just to integration with other applications. Much of the customization work of an application involves changing how the product actually functions as well.

Different users of a product may think of customization in different ways – changing menu items, integration with Office products, automating repetitive tasks – and all of these issues may be addressed by using VBA and automation. The following table lists five potential reasons for seeking to customize a product. Using VBA, all of them can be realized.

Category	What It Means	Rational System Architect Example
Modifying application behaviour	Modifying the way applications work to match a company's business rules and processes	On creating a symbol on a diagram the company naming standard rules are checked and the user flagged if they are broken.
Automating repetitive tasks	Combining sets of common manual tasks into a series of actions that can be executed over and over	Printing a user defined series of diagrams.
Extending application functionality	Adding features to an application that are not available out of the box	Automatic creation of a Process Map diagram derived from a Process Chart and its assigned roles and their Organizational Units.
Integrating with other applications	Controlling another application to exploit functionality not normally available	Producing an Excel Spreadsheet of Process versus Entity CRUD information.
Accessing corporate data	Exchanging data with remote databases and applications that aren't normally capable of	Automated import of information from diverse sources that cannot usually be imported directly.

	database access	

Combining Rational System Architect with VBA provides the user with a standard Integrated Development Environment (IDE) for creating these customizable solutions by incorporating all the programming developments provided by Microsoft, including IntelliSense and Microsoft forms.

Planning an Automated Solution with Rational System Architect

The uses for Automation are varied from reducing repetitive tasks to creating a fully integrated application. The next section looks at some of the potential changes that can be made to Rational System Architect.

Control Behavior

Rational System Architect will respond to certain events occurring during operation. These events can trigger code that may be used to automate certain tasks.

The events supported by Rational System Architect include Start up and Shutdown of the product, Encyclopedia Open and Encyclopedia Close, Diagram Open, Save and Close, Audit ID change and a number of symbol events. The symbol events include placing on a diagram, naming, connecting and disconnecting (with a line symbol) and deleting.

This functionality allows real-time operation of Rational System Architect to be modified using VBA.

Controlling appearance

Starting with Rational System Architect 10.1, users may customize menus in Rational System Architect, adding or subtracting commands from the existing menu structure. A user can add a command to run a macro, for example. Prior to Rational System Architect 10.1 (ie, 10.0 and before), the only way to add commands to menu items was through Rational System Architect VBA. For example, a menu containing items specific to a methodology could be updated during code execution. Those menu items would then only be shown for a specific diagram dependent on the methodology selected.

In either case, menu items and menu pop-ups may be added to menus and bitmaps assigned to these items. A common use of this technique is to add a macro to a menu item and supply a bitmap to represent it. This may also then be placed on a toolbar as well as a menu.

Information concerning how to adjust macros built to be used with menu items in Rational System Architect 10.0 or before, for Rational System Architect 10.1, are provided in the Conversion manual found on the IBM support site.

Automate tasks

A VBA macro can be created to report on information within the repository and enforce consistency checking. All objects and properties within the data dictionary may be created, read, updated or deleted based upon rules set within the code. Simple repetitive tasks like updating a value of one data dictionary object from another can therefore be automated.

Enforce controls

By using the event driven members of the Rational System Architect VBA model a predefined set of standards can be applied to the model in real time. This may be a naming standard or the completion of a mandatory field.

Interface between external applications

VBA can be used to import, export, read, create, modify, update, and delete model objects based upon values in other applications. Example functions include creating diagrams, symbols and definitions, building relationships between symbols and modifying the data dictionary properties.

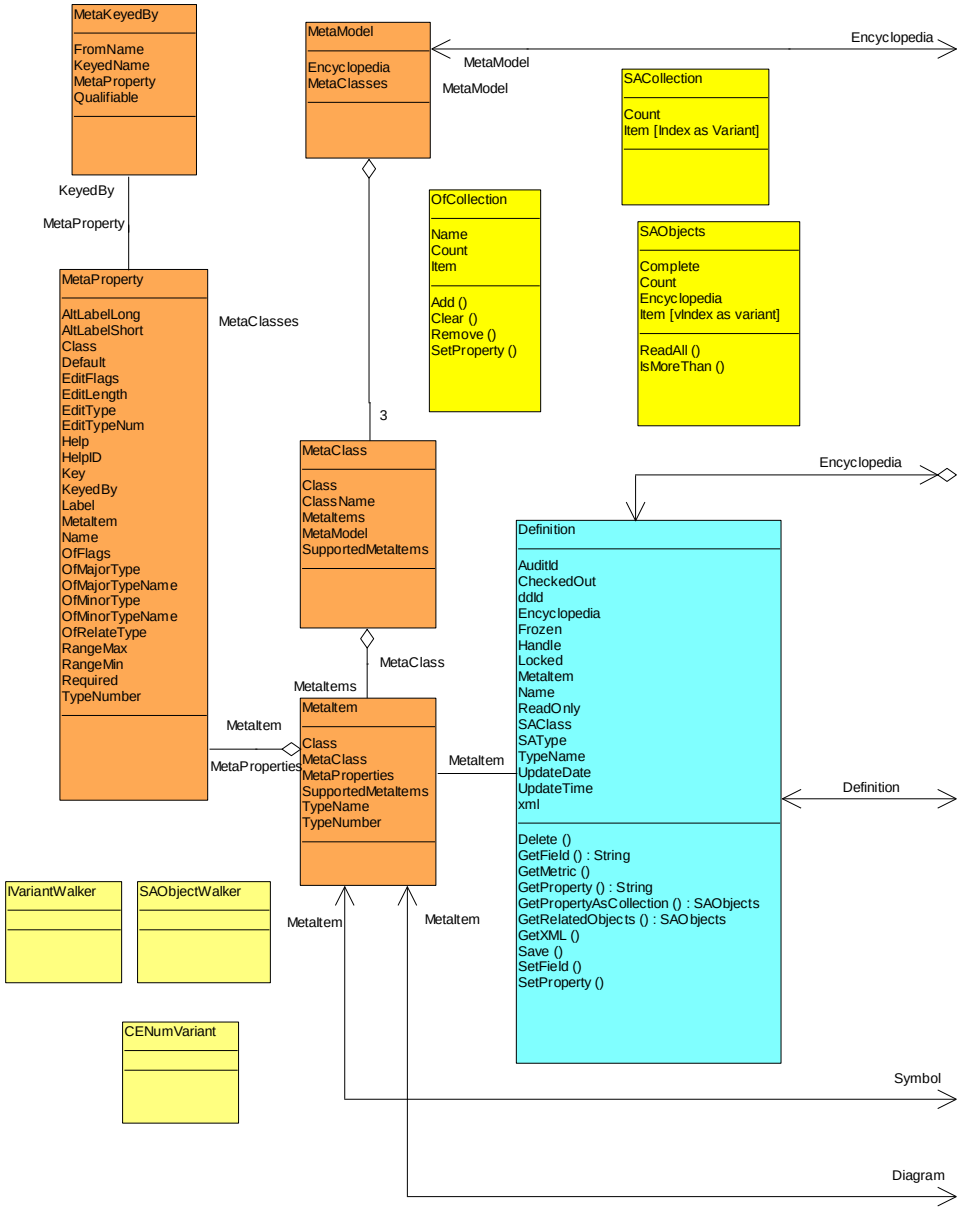
3

The Rational System Architect Object Model

The Objects available to VBA from Rational System Architect can be viewed through the Object Browser in the VBA editor. Each type of object is defined as a class, which contains a list of the supported properties and methods.

A much easier way of viewing the whole model is by using a class diagram. The following picture shows an extract from the Rational System Architect object model drawn in Rational System Architect using the UML Class diagram notation.

Topics in this chapter	Page


MetaKeyedBy
FromName
KeyedName
MetaProperty
Qualifiable
KeyedBy
MetaProperty
MetaProperty
AltLabelLong
AltLabelShort
Class
Default
EditFlags
EditLength
EditType
EditTypeNum
Help
HelpID
Key
KeyedBy
Label
MetaItem
Name
OfFlags
OfMajorType
OfMajorTypeName
OfMinorType
OfMinorTypeName
OfRelateType
RangeMax
RangeMin
Required
TypeNumber
MetaClasses
MetaModel
Encyclopedia
MetaClasses
MetaModel
MetaModel
Encyclopedia
SACollection
Count
Item [Index as Variant]
OfCollection
Name
Count
Item
Add ()
Clear ()
Remove ()
SetProperty ()
SAObjects
Complete
Count
Encyclopedia
Item [vIndex as variant]
ReadAll ()
IsMoreThan ()
3
MetaClass
Class
ClassName
MetaItems
MetaModel
SupportedMetaItems
MetaClass
MetaItems
MetaItem
MetaItem
Class
MetaClass
MetaProperties
SupportedMetaItems
TypeName
TypeNumber
MetaProperties
MetaItem
MetaItem
MetaItem
Encyclopedia
Definition
AuditId
CheckedOut
ddId
Encyclopedia
Frozen
Handle
Locked
MetaItem
Name
ReadOnly
SAClass
SAType
TypeName
UpdateDate
UpdateTime
xml
Delete ()
GetField () : String
GetMetric ()
GetProperty () : String
GetPropertyAsCollection () : SAObjects
GetRelatedObjects () : SAObjects
GetXML ()
Save ()
SetField ()
SetProperty ()
Definition
IVariantWalker
SAObjectWalker
CENumVariant
Symbol
Diagram


Encyclopedia
Application
ConnectString
FullName
MetaModel
Name
OpenObjectAsReadOnly
Path
xml
xmlex
bOpenLockedReadOnly ()
CloseUnlock ()
CreateDefinition () : Definition
CreateDiagram () : Diagram
GetAllDefinitions () : SAObjects
GetAllDiagrams () : SAObjects
GetCurrentDiagram ()
GetDefinitionById () : Definition
GetDiagramById () : Diagram
GetFilteredDefinitions () : SAObjects
GetFilteredDiagrams () : SAObjects
GetRelationMetric ()
GetXML ()
OpenEncyclopedia ()
OpenLock ()
SetXML ()
SetXMLEx ()
Application
Encyclopedia
Mouse Pointer
Visible
AssignBMPtoMacroItem ()
BrowserRefresh ()
CreatePopUpMenu () : long
InsertMacroItemInMenu ()
InsertMacroItemInMenuEx ()
InsertPopupMenuItemInMenu ()
InsertPopupMenuItemInMenuEx ()
Interface ()
OpenEncyclopedia ()
RemoveItemFromMenu ()
RemovePopUpMenu ()
ResetPanelBackGround ()
SetSeparatorBefore ()
WriteStatusLine ()
WriteStatusLineEx ()
Encyclopedia
MetaModel
Application
Encyclopedia
Encyclopedia
Encyclopedia
Encyclopedia
Symbol
ArrowAtEnd
ArrowAtStart
AuditId
ddId
Definition
Diagram
Encyclopedia
FillColor
FontColor
FromCardinality
Handle
LineStyle
MetaItem
Name
PenColor
PenStyle
SAClass
SAType
Selected
ToCardinality
TunnelAtEnd
TunnelAtStart
TypeName
UpdateDate
UpdateTime
XPos
XSize
YPos
YSize
ConnectFrom ()
ConnectTo ()
Delete ()
GetChildDiagrams ()
GetField () : String
GetMetric ()
GetProperty () : String
GetPropertyAsCollection () : SAObjects
GetRelatedObjects () : SAObjects
Save ()
SetField ()
SetProperty ()
Diagram
AuditId
CheckedOut
ddId
Encyclopedia
Frozen
Handle
Hidden
Lock
MetaItem
Name
Picture
ReadOnly
SAClass
SAType
TypeName
UpdateDate
UpdateTime
xml
CreateSymbol () : Symbol
Delete ()
GetAllSymbols () : SAObjects
GetField () : String
GetFilteredSymbols () : SAObjects
GetMetric ()
GetParentSymbol ()
GetProperty () : String
GetPropertyAsCollection () : SAObjects
GetRelatedObjects () : SAObjects
GetSymbolById () : Symbol
GetXML ()
Hide ()
Save ()
SetField ()
SetProperty ()
Show ()
Definition
Diagram
MetaItem
MetaItem

Object Model Classes

These are the Rational System Architect Object Model classes with several example uses listed for each one below.

Class	Example Uses
Application	Rational System Architect Events Menu manipulation
Encyclopedia	Create diagrams and definitions objects Retrieve diagram and definitions objects
Diagram	Create and retrieve symbols objects Diagram property manipulation
Symbol	Symbol property manipulation Symbol connection information Retrieve child diagram objects
Definition	Definition property manipulation Related definition manipulation
SACollection	Collection of Rational System Architect properties
SAObjects	Collection of Rational System Architect Objects (diagrams, symbols and definitions)
OfCollection	Collection of OneOf or ListOf diagrams or definitions.
MetaModel	Rational System Architect Meta Model Objects (Supported Diagram , Symbol

MetaClass MetaItem	and Definition types)
MetaKeyedBy	Rational System Architect Meta Model Keyed by Definitions (Keyed by definitions and their structure)
MetaProperty	Rational System Architect Meta Model Property Sets (Supported properties and their structure within each definition type)

4

The Application Class

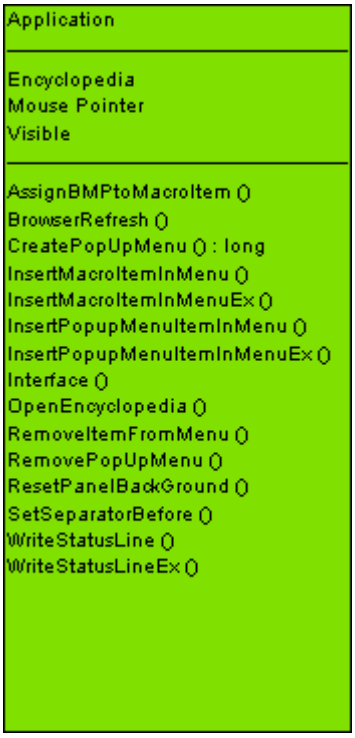
Topics in this chapter	Page
Attributes	4-3
Methods	4-5

Introduction

This is the Rational System Architect application object, through which the user interface can be controlled. It is the highest level in the object model.

An instance of the application object is instantiated as follows.

```
Dim oApplication As SA2001.Application
Set oApplication = New Application
```



Attributes

Encyclopedia

Purpose
This is the encyclopedia object, which facilitates access to the encyclopedia class attributes and methods.

Parameters
Read-only

Example
```
Dim oEncyclopedia As Encyclopedia
Set oEncyclopedia = oApplication.Encyclopedia
```

MousePointer

Purpose
This allows the user to control the type of mousepointer that is visible to the user when using the application.

Parameters
Data Type: Integer

Example
The current value of the mousepointer can be returned as follows;

```
Dim MouseValue as Integer
MouseValue  = oApplication.Mousepointer
```

To set a mousepointer to the “HourGlass” type use 11 as the value to set.

```
oApplication.Mousepointer  = 11
```

A complete list of permissible values can be found in the VB help files.

Visible

Purpose
Determines whether the application is running or not. Setting this to “False” will close the application down.

Parameter
Data Type: Boolean

Example
```
oApplication.Visible = False
```

Methods

AssignBMPtoMacroItem

Purpose
Associates a bitmap, for example a menu icon, with a VBA macro.

Syntax
```
Application Object.AssignBMPtoMacroItem MacroName,
  BMPFileName
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`MacroName`

Use: Required

Data Type: String

Any valid macro project

Syntax: “Project Name, Module Name, Subroutine Name()”

`BMPFileName`

Use: Required

Data Type: String

File Path & File Name of BMP (e.g. “C:\Windows\world.bmp”)

BrowserRefresh

Purpose
Refreshes the Rational System Architect browser. This will force any items added to the encyclopedia since the last refresh to be displayed.

Syntax

```
Application Object.BrowserRefresh
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

CreatePopUpMenu

Purpose

Creates a pop up menu for insertion into the user interface. A bitmap icon can be associated with the pop up menu.

Syntax

```
Application Object.CreatePopUpMenuPopUpName[, BMPFileName]
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`PopUpName`

Use: Required

DataType: String

Name of created popup menu

`BMPFileName`

Use: Optional

Data Type: String

File Path & File Name of BMP (e.g. “C:\Windows\world.bmp”)

InsertMacroItemInMenu

Purpose

Creates a menu item on the Rational System Architect menu that refers to an existing macro subroutine.

Syntax

```
Application Object.InsertMacroItemInMenu MacroName,
  MacroItemCaption, InMenuTitleCaption[,
  BeforeMenuItemCaption]
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`MacroName`

Use: Required

Data Type: String

Any valid macro project

Syntax: “Project Name, Module Name, Subroutine Name()”

`MacroItemCaption`

Use: Required

Data Type: String

Name of macro item that will be inserted in an SA menu

`InMenuTitleCaption`

Use: Required

Data Type: String

Name of existing SA popup menu where macro item is being placed

`BeforeMenuItemCaption`

Use: Optional

Data Type: String

Name of existing SA menu item the macro item is being placed before

Note: If not specified, macro is placed at the bottom of the popup menu.

InsertMacroItemInMenuEx

Purpose

Creates a menu item on the Rational System Architect menu that refers to an existing macro subroutine. This method is an extension of the InsertMacroItemInMenu method.

Syntax

```
Application Object.InsertMacroItemInMenuEx MacroName,
  MacroItemCaption, InMenuTitleCaption[,
  BeforeMenuItemCaption[, Tag[, bAfterSepartor]]]
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`MacroName`

Use: Required

Data Type: String

Any valid macro project

Syntax: “Project Name, Module Name, Subroutine Name()”

`MacroItemCaption`

Use: Required

Data Type: String

Name of macro item that will be inserted in an SA menu

`InMenuTitleCaption`

Use: Required

Data Type: String

Name of existing SA popup menu where macro item is being placed

`BeforeMenuItemCaption`

Use: Optional

Data Type: String

Name of existing SA menu item the macro item is being placed before

Note: If not specified, macro is placed at the bottom of the popup menu.

`Tag`

Use: Optional

Data Type: String

This allows multiple menu items to have the same name and refer to the same subroutine by giving each menu item a unique tag. When one of the menu items is called, its tag tells the subroutine which part of its code to execute. For example, a user can write a macro that creates different Word Reports. Instead of having to write separate subroutines for each type of Word Report, he or she can write all the code in one subroutine and specify different tags in each menu item that point to different functions within the code.

`bAfterSepartor`

Use: Optional

DataType: Boolean

Only used when the existing menu item, which the user is placing the macro item in front of, has a separator line before it. If entered as True, then the macro item is placed after the separator line. If entered as False or left blank, then the macro item is automatically placed before the separator line.

InsertPopUpMenuItemInMenu

Purpose

Creates a pop up menu item into an existing Rational System Architect menu item.

Syntax

```
Application Object.InsertPopupMenuItemInMenu PopUpName,
  InMenuTitleCaption[, BeforeTitleCaption]
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`PopUpName`

Use: Required

DataType: String

Name of created popup menu

`InMenuTitleCaption`

Use: Required

Data Type: String

Name of existing SA popup menu where new popup menu is being placed

`BeforeMenuItemCaption`

Use: Optional

Data Type: String

Name of existing SA menu item the new popup menu is being placed before

Note: If not specified, new popup menu is placed at the bottom of the existing popup menu.

InsertPopUpMenuItemInMenuEx

Purpose

Creates a pop up menu item in to an existing Rational System Architect menu item. This method is an extension of the InsertPopUpMenuItemInMenu method.

Syntax

```
Application Object.InsertPopupMenuItemInMenuEx PopUpName,
  InMenuTitleCaption[, BeforeTitleCaption[,
  bAfterSeparator]]
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`PopUpName`

Use: Required

DataType: String

Name of created popup menu

`InMenuTitleCaption`

Use: Required

Data Type: String

Name of existing SA popup menu where new popup menu is being placed

`BeforeMenuItemCaption`

Use: Optional

Data Type: String

Name of existing SA menu item the new popup menu is being placed before

Note: If not specified, new popup menu is placed at the bottom of the existing popup menu.

`bAfterSepartor`

Use: Optional

DataType: Boolean

Only used when the existing menu item, which the user is placing the macro item in front of, has a separator line before it. If entered as True, then the macro item is placed after the separator line. If entered as False or left blank, then the macro item is automatically placed before the separator line.

Interface

Purpose

This method is rarely used but can call an instance of an interface using a text string rather than an explicit reference.

Example

```
Dim sa As Application
Set sa = New Application
```

```
Dim ob As Object
Set ob = sa.Interface("ISAIMF")
```

OpenEncyclopedia

Purpose
Opens an existing Rational System Architect Encyclopedia.

Syntax
```
Application Object.OpenEncyclopedia(EncyclopediaPath)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`EncyclopediaPath`

Use: Required

Data Type: String

File path of existing encyclopedia

The (EncyclopediaPath) is a udl file. The udl files are indirectly created in the following path C:\Document and Settings\<user name>\Local Settings\Application Data\Telelogic\System Architect\Temp UDL files. These udl files will be named something like SA_563.udl. Rational System Architect must be opened to view this path.

OpenEncyclopediaUsingConnectionString

Purpose
Opens an existing Rational System Architect Encyclopedia using a connection string.

Syntax
```
Application Object.OpenEncyclopediaUsingConnectionsString
  (strConnection)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

strConnection

Use: Required

Data Type: String

strConnection is a string that is the contents of the UDL file.

For example:

```
SA2001.OpenEncyclopediaUsingConnectionString
  ("Provider=SQLOLEDB.1;Integrated
  Security=SSPI;InitialCatalog=DoDAFABM;Data
  Source=SUZANNES\TLOGICSA106")
```

OpenEncyclopediaUsingDisplayName

Purpose

Opens an existing Rational System Architect Encyclopedia using its display name

Syntax

Application Object.OpenEncyclopediaUsingDisplayName(strDisplayName)

Application Object

Use: Required

Data Type: Object

Any instantiated Application class

strDisplayName

Use: Required

Data Type: String

strDisplayName is the name as it is shown on the SA caption bar i.e. connection-name(encyc-name).

Example:

Sa2001.OpenEncyclopediaUsingDisplayName “Local Server SUZANNESTLOGICSA 106(Samples)”

RemoveItemFromMenu

Purpose

This method removes a menu item from a named Rational System Architect Menu item or pop up menu.

Syntax

```
Application Object.RemoveItemFromMenu ItemCaption,
  FromMenuTitleCaption
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`ItemCaption`

Use: Required

Data Type: String

Name of menu item being removed from existing popup menu

`FromMenuTitleCaption`

Use: Required

Data Type: String

Name of existing popup menu from which menu item is being removed

RemovePopUpMenu

Purpose

The specified pop up menu will be removed from the Rational System Architect menu system.

Syntax

```
Application Object.RemovePopUpMenu(PopUpName)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`PopUpName`

Use: Required

DataType: String

Name of popup menu being removed

ResetPanelBackGround

Purpose

Resets the background color of the status bar panel.

Syntax

```
Application Object.ResetPanelBackGround(Panel)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`Panel`

Use: Required

Data Type: Long

Panels are the 'panes' or sections of the status bar - 1 on the left through to 4 on the right (2&3 only appear when a symbol is selected).

SetSeparatorBefore

Purpose
Places a separator bar before a menu item in a specified menu.

Syntax
```
Application Object.SetSeparatorBefore ItemCaption,
  FromMenuTitleCaption, bHasSeparator)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`ItemCaption`

Use: Required

Data Type: String

Name of menu item the separator bar will be placed before.

`FromMenuTitleCaption`

Use: Required

Data Type: String

Name of existing SA popup menu in which the separator is being placed.

`bHasSeparator`

Use: Required

Data Type: Boolean

Sets a true or false value on whether the separator bar is present.

WriteStatusLine

Purpose
Allows short messages to be relayed to the user to keep them informed whilst code is being executed on the status bar of Rational System Architect (bar in the bottom left hand corner).

Syntax

```
Application Object.WriteStatusLine(TextToShow)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`TextToShow`

Use: Required

Data Type: String

Text that will be shown on status bar.

WriteStatusLineEx

Purpose

Allows short messages to be relayed to the user to keep them informed whilst code is being executed on the status bar of Rational System Architect (bar in the bottom left hand corner). This is an extension of the WriteStatusLine method.

Syntax

```
Application Object.WriteStatusLineEx(Panel, TextToShow,
  BackColor, ForeColor)
```

`Application Object`

Use: Required

Data Type: Object

Any instantiated Application class

`Panel`

Use: Required

Data Type: Long

Panels are the 'panes' or sections of the status bar - 1 on the left through to 4 on the right (2&3 only appear when a symbol is selected).

`TextToShow`

Use: Required

Data Type: String

Text that will be shown on status bar.

`BackColor`

Use: Required

Data Type: Long

Background color of the status bar

`ForeColor`

Use: Required

Data Type: Long

Foreground color of the status bar

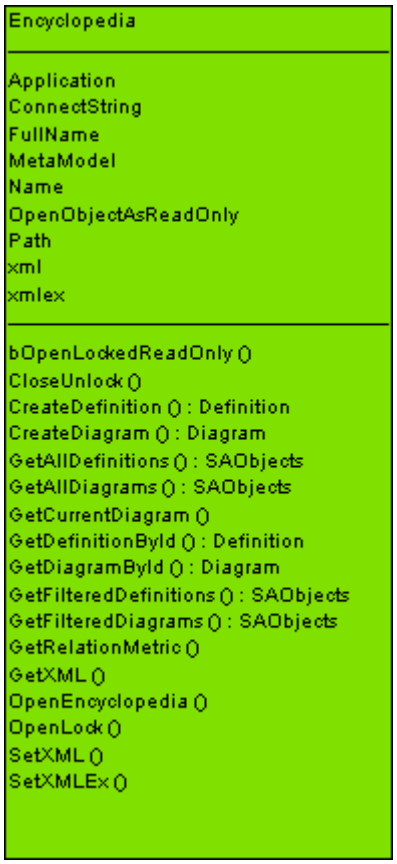
5

The Encyclopedia Class

Introduction

This is the encyclopedia object. This enables access to the encyclopedia's attributes and methods as detailed below.

```
Dim oApplication As SA2001.Application, oEncyclopedia
  As Encyclopedia
Set oApplication = New Application
Set oEncyclopedia = oApplication.Encyclopedia
```



Attributes

Application

Purpose
The application object returns the parent application object of the current encyclopedia object.

Parameters
Read-only

ConnectString

Purpose
The information required to connect to an encyclopedia.

Parameters:
Data Type: String

Read-only

FullName

Purpose
The name of the current encyclopedia including the full path.

Parameters
Read-only

Data Type: String

MetaModel

Purpose

This is the MetaModel Class. If facilitates access to all the MetaModel attributes.

Parameters
Read-only

Name

Purpose
Returns the name of the current encyclopedia.

Parameters
Read-only

Data Type: String

OpenObjectsAsReadOnly

Purpose
Sets whether all objects from the SA Object model are to be opened as Read-only.

Parameters
DataType: Boolean

Path

Purpose
This is the path of the current encyclopedia.

Parameters
Data Type: String

Read-only

Xml

Purpose
The xml string of the encyclopedia. Manipulated by the GetXML and SetXML methods.

Parameters
Data Type: String

XmlEx

Purpose
The xml string of the encyclopedia. Manipulated by the SetXMLEx method.

Parameters
Data Type: String

Methods

bOpenLockedReadOnly

Purpose
This method returns True if the OpenObjectsAsReadOnly attribute has been set to True, or the encyclopedia was opened read-only.

CloseUnlock

See OpenLock below.

CreateDefinition

Purpose
Creates an instance of the definition class with a specified definition name and definition type.

Syntax
```
Encyclopedia Object.CreateDefinition(Name, SAType)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`Name`

Use: Required

Data Type: String

Name of new definition

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect definition that is being created (e.g. DFXPROCESS or 3)

Note: Refer to the DEFNS.BAS file in the Rational System Architect directory for a complete listing of all SA definitions and their internal constant names and numbers.

Note: In order to successfully create a SA definition, you must invoke the definition's Save method, or else the new definition will be deleted when the encyclopedia is closed.

CreateDiagram

Purpose

Creates an instance of the diagram class with a specified diagram name and diagram type.

Syntax

```
Encyclopedia Object.CreateDiagram(Name, SAType)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`Name`

Use: Required

Data Type: String

Name of new diagram

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect diagram that is being created (e.g. GTCATPROCESSFLOW or 89)

Note: Refer to the DIAGRAMS.BAS file in the Rational System Architect directory for a complete listing of all SA diagrams and their internal constant names and numbers.

GetAllDefinitions

Purpose
This will return all definitions in the encyclopedia as a collection of definitions.

Rules
An SAObjects variable must be dimensioned and set as a collection of definitions.

Example

```
Dim oCollectionofDefinitions As SAObjects

Set oCollectionofDefinitions =
  oEncyclopedia.GetAllDefinitions

Call oCollectionofDefinitions.ReadAll
```

The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetAllDefinitions should be used in conjunction with either ReadAll or IsMoreThan methods.

GetAllDiagrams

Purpose
This will return all diagrams in the encyclopedia as a collection of diagrams.

Rules
An SAObjects variable must be dimensioned and set as a collection of diagrams.

Example

```
Dim oCollectionofDiagrams As SAObjects

Set oCollectionofDiagrams =
  oEncyclopedia.GetAllDiagrams

Call oCollectionofDiagrams.ReadAll
```

The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetAllDiagrams should be used in conjunction with either ReadAll or IsMoreThan methods.

GetCurrentDiagram

Purpose
This method returns the currently open diagram as a diagram object.

Rules
A diagram object must be dimensioned and set as the current open diagram. See example below.

Example
```
Dim OCurrentDiagram As Diagram
Set OCurrentDiagram = oEncyclopedia.GetCurrentDiagram
```

GetDefinitionById

Purpose
This method will return a definition as a definition object from its specified identity.

Syntax
```
Encyclopedia Object.GetDefinitionById(Id)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`Id`

Use: Required

Data Type: Long

All definitions stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier.

Example

```
Dim oDefinition As Definition
Set oDefinition = oEncyclopedia.GetDefinitionById(12)
```

GetDiagramById

Purpose

All diagrams stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier. This method will return a diagram as a diagram object from its specified identity.

Syntax

```
Encyclopedia Object.GetDiagramById(Id)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`Id`

Use: Required

Data Type: Long

All diagrams stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier.

Example

```
Dim oDiagram  As Diagram
Set oDiagram = oEncyclopedia.GetDiagramById(2)
```

GetFilteredDefinitions

Purpose

Returns a filtered definition collection of an encyclopedia.

Parameters

Data Type: SAObjects

Syntax

```
Encyclopedia Object.GetFilteredDefinitions(WildCardName,
  SAType)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`WildCardName`

Use: Required

Data Type: String

Filter criteria (e.g. “C*” = all definitions starting with “C”)

Note: Wild Card Search is case sensitive.

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect definition that is being retrieved (e.g. DFXPROCESS or 3)

Note: Refer to the DEFNS.BAS file in the Rational System Architect directory for a complete listing of all SA definitions and their internal constant names and numbers

Example

The following returns all process definitions beginning with “C”.

```
Dim oCollectionofDefinitions As SAObjects

Set oCollectionofDefinitions =
  oEncyclopedia.GetFilteredDefinitions("C*",
  DFXPROCESS)

Call oCollectionofDefinitions.ReadAll
```

The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetFilteredDefinitions should be used in conjunction with either ReadAll or IsMoreThan methods.

GetFilteredDiagrams

Purpose
Returns a filtered diagram collection of an encyclopedia.

Parameters
Data Type: SAObjects

Syntax
```
Encyclopedia Object.GetFilteredDiagrams(WildCardName,
  SAType)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`WildCardName`

Use: Required

Data Type: String

Filter criteria (e.g. “C*” = all diagrams starting with “C”)

Note: Wild Card Search is case sensitive.

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect diagram that is being retrieved (e.g. GTCATPROCESSFLOW or 89)

Note: Refer to the DIAGRAMS.BAS file in the Rational System Architect directory for a complete listing of all SA diagrams and their internal constant names and numbers.

Example
The following returns all Gane & Sarson Diagrams beginning with “Pr”

```
Dim oCollectionofDiagrams As SAObjects
```

```
Set oCollectionofDiagrams =
  oEncyclopedia.GetFilteredDiagrams("Pr*", GTDFDGS)

Call oCollectionofDiagrams.ReadAll
```

The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetFilteredDiagrams should be used in conjunction with either ReadAll or IsMoreThan methods.

GetRelationMetric

Purpose

Gets relationship information or behavior between two Rational System Architect objects in the encyclopedia.

Syntax

```
Encyclopedia Object.GetRelationMetric SAObject1,
  SAObject2, Relation, Depth, Metric, FieldType[,
  NbrChars[, NbrDec]]
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`SAObject1`

Use: Required

Data Type: Object

One of two required Rational System Architect objects that are needed to execute the relation metric.

`SAObject2`

Use: Required

Data Type: Object

One of two required Rational System Architect objects that are needed to execute the relation metric.

`Relation`

Use: Required

Data Type: RELATETYPE

The relationship type that exists between the two parameterized SA objects above. See Chapter 16 for a full list of all Rational System Architect relationship types and their descriptions.

`Depth`

Use: Required

Data Type: Long

The number of relationships between the two parameterized SA objects above. For example, if Object 1 is a Data Structure that contains Object 2, which is a Data Element, then the 'Depth' between the two objects is 1.

`Metric`

Use: Required

Data Type: RELATIONMETRIC

Relation Metric, see below for a complete list of all relation metrics.

`FieldType`

Use: Required

Data Type: FLDTYPE

Field Type, see Chapter 17 for a complete list of Rational System Architect field types.

`NbrChars`

Use: Optional

Data Type: Long

The number of characters that SA will return that will appear before the decimal point.

`NbrDec`

Use: Optional

Data Type: Long

The number of characters that SA will return that will appear after the decimal point.

GetXML

Purpose
Exports the encyclopedia's XML string into a valid .xml file.

Syntax
```
Encyclopedia Object.GetXML strXML, bToFile
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`StrXML`

Use: Required

Data Type: String

When `bToFile` is set to True, then this is a valid xml file name to which SA will export the encyclopedia's xml. When `bToFile` is set to false, strXML acts as the xml string.

`bToFile`

Use: Required

Data Type: Boolean

If True, then the method will create the file named in the strXML parameter. If False, then the method will populate strXML with the encyclopedia xml string.

OpenLock…CloseUnlock Statement

Purpose
The OpenLock and CloseUnlock methods control the lock status of the current Rational System Architect encyclopedia. This determines whether the encyclopedia is locked for read-only, read-write or update access while VBA operations are taking place.

If an OpenLock is executed in a particular mode then a CloseUnlock must be executed in the same mode later in the code.

OpenLock and CloseUnlock methods can be operated multiple times in code if different levels of locking are required on the encyclopedia.

If OpenLock and CloseUnlock methods are not executed in the VBA code then Rational System Architect will perform it's own locking when required each time an Object Model method is issued. This can affect the performance of the macro.

The two methods return a Boolean indicating whether the call was successful or not.

Syntax

```
Encyclopedia Object.OpenLock(LockMode)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`LockMode`

Use: Required

Data Type: EncyLockMode

Lock status of current Rational System Architect Encyclopedia.

EncyLockMode	Meaning
NETOPENREAD	Read-only
NETOPENREADWRITE	Read-write
NETOPENUPDATE	Update access while VBA applications are taking place.

Example

```
Dim sa As Application
Set sa = New Application
sa.Encyclopedia.OpenLock NETOPENREAD
    ' execute SA Code here
sa.Encyclopedia.CloseUnLock NETOPENREAD
Set sa = Nothing
```

SetXML

Purpose
Imports an .xml file into the encyclopedia.

Syntax
```
Encyclopedia Object.SetXML(strXML, bFromFile, bValidate)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`StrXML`

Use: Required

Data Type: String

When bFromFile is set to True, then this is a valid .xml file name, from which SA is importing the XML code. When bFromFile is set to false, then this is the encyclopedia XML string.

`bFromFile`

Use: Required

Data Type: Boolean

If True, then the method will import XML from the file named in the strXML parameter. If False, then the method will populate strXML with the encyclopedia XML string.

`bValidate`

Use: Required

Data Type: Boolean

If True then the xml string will be validated by the parser.

SetXMLEx

Purpose

Syntax

```
Encyclopedia Object.SetXMLEx(strXML, ICollision,
  bFromFile, bValidate)
```

`Encyclopedia Object`

Use: Required

Data Type: Object

Any instantiated Encyclopedia class

`StrXML`

Use: Required

Data Type: String

When bFromFile is set to True, then this is a valid .xml file name, from which SA is importing the XML code. When bFromFile is set to false, then this is the encyclopedia XML string.

`ICollision`

Use: Required

Data Type: Long

Collision Options	Description
0	Never overwrite an existing definition or diagram
1	If the definition exists, get all properties for it, delete it, recreate it, repopulate the properties.
2	Update single fields when data supplied
3	Update single fields - clear field if no data
256	Always replace existing diagram

`bFromFile`

Use: Required

Data Type: Boolean

If True, then the method will import XML from the file named in the strXML parameter. If False, then the method will populate strXML with the encyclopedia XML string.

`bValidate`

Use: Required

Data Type: Boolean

If True then the XML string will be validated by the parser.

Relation Metrics

Relation metrics differ from diagram, symbol, and definition metrics in that they are pieces of internal functionality that retrieve information on the relationship between two Rational System Architect Objects in the encyclopedia. For each relation metric, it is necessary to declare which two objects should be examined and what relationship exists between them. Depending on which relation metric is being utilized only certain Rational System Architect objects with specific relationships are valid.

To access these relation metrics, the user must invoke the GetRelationMetric method in the Encyclopedia class. There exists a directory of all the relation metrics in the RELATIONMETRIC enumerated list in the SA Object Browser. The following is a table of all the available relation metrics and their descriptions and parameters.

Relation Metric	Description	Parameters
RELMETCRUD	Returns the combination of CRUD letters (Create, Read, Update, Delete) appear next to the Process name in the Data Store as a string.	SAObjects: Data Store, Process
RELMETDEPTH	Returns the number of 'Uses' relationships between two objects as a numeric.	SAObjects: Must have at least one 'Uses' or 'Used By' relationship.
RELMETICOMROLE	Returns the relationship role (input, control, output, mechanism, or boundary) between an ICOM arrow and its connected Function/Activity symbol as a string.	SAObjects: ICOM Arrow, Function/Activity RelTypes: RELCONNSTART, RELSTARTAT, RELCONNEND, RELENDAT
RELMETINPUT	Checks whether a flow symbol "flows" into another symbol or diagram. Returns Boolean field.	SAObjects: Flow symbol, diagram or node symbol RelType: RELCONNSTART, RELSTARTAT, RELCONNEND, RELENDAT, RELDIAGRAMCON, RELCONDIAGRAM

Relation Metric	Description	Parameters
RELMETOUTPUT	Checks whether a flow symbol “flows” out of another symbol or diagram. Returns Boolean field.	SAObjects: Flow symbol, diagram or node symbol RelType: RELCONNSTART, RELSTARTAT, RELCONNEND, RELENDAT, RELDIAGRAMCON, RELCONDIAGRAM
RELMETSTATETABLE	Checks whether the name of the event attached to a output transition line, which is connected to a state, is mentioned in the state definition dialog. If True, returns State name.	SAObjects: Shlaer State, Shlaer Transition line RelType: RELCONNEND, RELCONNSTART

6

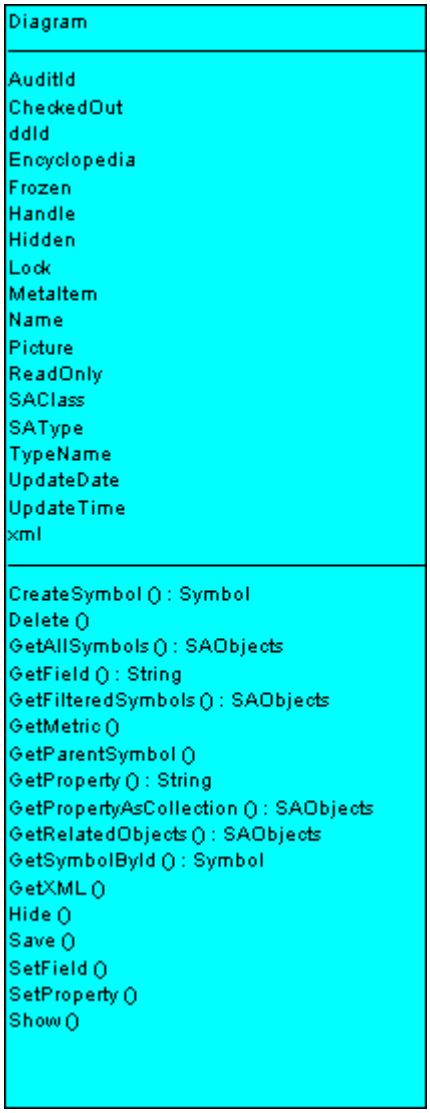
The Diagram Class

Introduction

This is an instance of a diagram contained in the encyclopedia.

To return the currently active diagram as an object use:

```
Dim oApplication As
  SA2001.Application

Dim oDiagram As Diagram

Set oApplication = New
  Application

Set oDiagram =
  oApplication.oEncyclopedia.
  GetCurrentDiagram
```



Attributes

AuditID

Purpose
All items diagrams stored in Rational System Architect are tagged with the identity of the person who created or last modified the diagram; the identity is tagged to the diagram as an AuditId.

Parameters
Data Type: String

Read-only

CheckedOut

Purpose
When set to True diagram becomes read-only to everyone but the AuditID that checked it out.

Parameters
Data Type: Boolean

ddID

Purpose
All diagrams stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier. This method will return the identity of a diagram.

Parameters
Data Type: Long

Encyclopedia

Purpose
Facilitates access with the parent encyclopedia's attributes and methods.

Parameters
Read-only

Frozen

Purpose
User must have Freezing privileges in order to set this attribute. When set to True, diagram becomes read-only to everyone, including the AuditID that froze the diagram.

Parameter
Data Type: Boolean

Handle

Purpose
This is the memory handle of the diagram only available at run-time. This handle is not unique and is rarely the same whenever accessed.

Parameters
Data Type: Long

Read-only

Example

```
Dim oDiagram as Diagram, Handle As Long
Set oDiagram = oEncyclopedia.GetCurrentDiagram
Handle = oDiagram.Handle
```

Hidden

Purpose
Returns a “True” or “False” to indicate whether a diagram is closed or open.

Parameters
Data Type: Boolean

Read-only

Locked

Purpose
Returns a “True” or “False” value to indicate whether a diagram is locked or not locked, i.e., in use by a user.

Parameters
Data Type: Boolean

Read-only

MetaItem

Purpose
Facilitates access with the MetaItem class and its attributes.

Parameters
Read-only

Name

Purpose
The name of the specified diagram object.

Parameters

Data Type: String

Read-only

Picture

Purpose
After a diagram is saved, Rational System Architect creates a Windows metafile (.wmf) of the diagram, which is saved in the encyclopedia directory. This attribute allows the user to access the stdPicture attributes and methods, an OLE Automation object that holds data on the picture's contents.

Parameters
Data Type: stdPicture

Read-only

ReadOnly

Purpose
Returns "True" if diagram has been opened as Read-only.

Parameters
Data Type: Boolean

Read-only

SAClass

Purpose
The class type of the diagram. Also known as the Major Type Number.

Parameters
Data Type: Long

Read-only

Note: this will always return "1" for a diagram.

SAType

Purpose
The constant integer of the diagram. All diagrams in Rational System Architect have a unique numerical constant identifier.

Parameters
Data Type: Long

Read-only

TypeName

Purpose
The type of the diagram as a string, e.g. “Entity Relation”.

Parameters
Data Type: String

Read-only

UpdateDate

Purpose
The date the diagram was last modified.

Parameters
Data Type: String

Read-only

UpdateTime

Purpose
The time the diagram was last modified.

Parameters
Data Type: String

Read-only

xml

Purpose
The xml string of the diagram. Manipulated by the GetXML method.

Parameters
Data Type: String

Read-only

Methods

CreateSymbol

Purpose
Creates an instance of the Symbol class with a particular name and type.

Syntax
```
Diagram Object.CreateSymbol(Name, SAType)
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class, to which the symbol will be added.

`Name`

Use: Required

Data Type: String

Name of new symbol

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect symbol that is being created (e.g. ETCATELEMBUSPROC or 445)

Note: Refer to the SYMBOLS.BAS file in the Rational System Architect directory for a complete listing of all SA symbols and their internal constant names and numbers.

Note: In order to successfully create a SA symbol on a diagram, you must invoke the diagram class' Save method, or else the new symbol will be deleted from the diagram when the encyclopedia is closed.

Delete

Purpose
Deletes a diagram specified by a diagram object.

GetAllSymbols

Purpose
This method returns all symbols contained in the specified diagram object as a SAObjects collection. The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetAllSymbols should be used in conjunction with either ReadAll or IsMoreThan methods.

Rules
An SAObjects variable must be dimensioned and set as a collection of symbols. See example below.

Example
```
Dim oDiagram as Diagram, oCollectionofSymbols As
  SAObjects
Set oCollectionofSymbols = oDiagram.GetAllSymbols
Call oCollectionofSymbols.ReadAll
```

GetField

Purpose
These are characteristics of the diagram, such as “Symbol Grid Size”, “Line Grid Size”, “Level Number” of which some can be set such as “Diagram Name” and some that cannot such as “Diagram Type”.

Syntax
```
Diagram Object. GetField FieldID
```

```
Diagram Object
```

Use: Required

Data Type: Object

Any instantiated Diagram class

`FieldID`

Use: Required

Data Type: DGMFLD

Diagram Field, see below for a complete list of all diagram fields.

GetFilteredSymbols

Purpose

To filter the symbols contained in a diagram, specify the filter criteria as a wildcard for the first argument and specify the type of symbol of interest as the second argument, part of which can also be "". This will return an SAObjects collection. The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetFilteredSymbols should be used in conjunction with either ReadAll or IsMoreThan methods.

Parameters

Data Type: SAObjects

Syntax

`Diagram Object.`**`GetFilteredSymbols`**`(WildCardName, SAType)`

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`WildCardName`

Use: Required

Data Type: String

Filter criteria (e.g. "C" = all symbols starting with "C")

Note: Wild Card Search is case sensitive.

`SAType`

Use: Required

Data Type: Long

Type of Rational System Architect symbol that is being created (e.g. ETCATELEMBUSPROC or 445)

Note: Refer to the SYMBOLS.BAS file in the Rational System Architect directory for a complete listing of all SA symbols and their internal constant names and numbers.

Example

This example will return all “Entity” symbols contained in the diagram starting with the letter “P”.

```
Dim oDiagram as Diagram, oCollectionofSymbols As
  SAObjects

Set oCollectionofSymbols =
  oDiagram.GetFilteredSymbols("P", ETECACTIVITY)

Call oCollectionofSymbols.ReadAll
```

Get Metric

Purpose

Calls certain lists, calculations, and pieces of internal functionality pertaining to diagrams.

Syntax

```
Diagram Object.GetMetric Metric[, FieldType[, NbrChars[,
  NbrDec]]]
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`Metric`

Use: Required

Data Type: DIAGRAMMETRIC

Diagram Metric, see below for a complete list of all diagram metrics.

`FieldType`

Use: Optional

Data Type: FLDTYPE

Field Type, see Chapter 17 for a complete list of Rational System Architect field types.

`NbrChars`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many characters are to be returned before the decimal point.

`NbrDec`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many numbers are to be returned after the decimal point.

GetParentSymbol

Purpose

A Diagram can be the child of a Parent symbol as in a Data Flow diagram. This method returns the parent symbol object for the specified diagram object.

GetProperty

Purpose

This returns the property content for any given diagram property.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in an Elementary Business Process of a Process Chart diagram, the "Locations" property is actually a rename – the real property is “Location Types”. You would only know this if you looked up the definition of an Elementary Business Process in saprops.cfg and saw that the property is actually called “Location Types” but has been ‘labeled’ “Locations”:

Property "Location Types" { Edit Listof "Location" Label "Locations" LENGTH 2000
HELP "Supporting Location Types (Matrix)" READONLY }

Syntax

```
Diagram Object.GetProperty Name
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

GetPropertyAsCollection

Purpose

Some properties define relationships with other properties. For example a Process Chart refers to a Process Thread through its “Process Thread” property. This method returns a collection of OneOf and ListOf diagrams or definitions. See Chapter 14 for more information on OneOf and ListOf property types.

Parameters

Data Type: OfCollection

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in an Elementary Business Process of a Process Chart diagram, the "Locations" property is actually a rename – the real property is “Location Types”. You would only know this if you looked up the definition of an Elementary Business Process in saprops.cfg and saw that the property is actually called “Location Types” but has been ‘labeled’ “Locations”:

Property "Location Types" { Edit Listof "Location" Label "Locations" LENGTH 2000
HELP "Supporting Location Types (Matrix)" READONLY }

Syntax

```
Diagram Object.GetPropertyAsCollection(PropName)
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`PropName`

Use: Required

Data Type: String

The name of the property as seen in the saprops.cfg

Example

```
Dim i As Long, DiagId As Long

i = 0

Do While sa.Encyclopedia.GetFilteredDiagrams("",
  GTCATPROCESSFLOW).IsMoreThan(i)

    i = i + 1

    Dim ThreadColl As OfCollection

    Set SADiag =
  sa.Encyclopedia.GetFilteredDiagrams("",
  GTCATPROCESSFLOW).Item(i)

    Set ThreadColl =
  SADiag.GetPropertyAsCollection("Process Thread")

Loop
```

GetRelatedObjects

Purpose

This method returns an SAObjects collection of related objects to the current diagram object.

Syntax

```
Diagram Object.GetRelatedObjects(RelType)
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`RelType`

Use: Required

Data Type: RELATETYPE

SA Relationship, see Chapter 16 for a complete list of all relationships.

GetSymbolById

Purpose

All diagrams stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier. This method will return a symbol as an object from its identity.

Syntax

```
Diagram Object.GetSymbolById(Id)
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`Id`

Use: Required

Data Type: Long

All symbols stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier.

Example

```
Dim oSymbol As Symbol
Set oSymbol = oDiagram.GetSymbolById(12)
```

GetXML

Purpose
Exports the Diagram's XML string into a valid .xml file.

Syntax
```
Diagram Object.GetXML(strXMLTextOut)
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`StrXMLTextOut`

Use: required

Data Type: String

A valid .xml file to which SA will export the diagram's XML string.

Hide

Purpose
Used to close an instance of a diagram that is currently open.

Syntax
```
Call oDiagram.Hide
```

Save

Purpose
Used to save an instance of a diagram.

Syntax
```
Call oDiagram.Save
```

SetField

Purpose
SetField allows a Diagram Field to be set with a specified value.

Syntax
```
Diagram Object.SetField FieldID, value
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`FieldID`

Use: Required

Data Type: DGMFLD

Diagram Field, see below for a complete list of all Diagram fields.

`Value`

Use: Required

Data Type: String

Value of diagram field

SetProperty

Purpose
The setting of a diagram property value requires the name of the property as the first argument and the value to set as the second. The property names are found in the saprops.cfg and usrprops.txt files.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in an Elementary Business Process of a Process Chart diagram, the "Locations" property is actually a rename – the real property is

“Location Types”. You would only know this if you looked up the definition of an Elementary Business Process in saprops.cfg and saw that the property is actually called “Location Types” but has been ‘labeled’ “Locations”:

Property "Location Types" { Edit Listof "Location" Label "Locations" LENGTH 2000
HELP "Supporting Location Types (Matrix)" READONLY }

Syntax

```
Diagram Object.SetProperty Name, value
```

`Diagram Object`

Use: Required

Data Type: Object

Any instantiated Diagram class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

`Value`

Use: Required

Data Type: String

Value of diagram property

Show

Purpose

This method will open the diagram on a Rational System Architect screen.

Syntax

```
Call oDiagram.Show
```

Diagram Fields

The diagram field property can contain a number of properties held about the diagram. They typically contain information that a user cannot enter directly but is derived during normal use. Within the Object Model there is an enumerated type called *DGMFLD*. This is passed as a parameter to the operation **GetField(FieldID as *DGMFLD*)** and **SetField(FieldID as *DGMFLD*, Value as String)**. This enables the VBA programmer to both read and update low level diagram fields.

DGMFLD constant	Description	Data Type
DIAGFLD_BBORDER	Toggles the Report Border check box in the Page Setup window. This enables the user to place a border around the report.	"O" = unchecked "1" = checked
DIAGFLD_BDGMBORDER	Selects a border from the diagram border drop-list in the Page Setup window.	"0" = no form/ no border "1" = simple border
DIAGFLD_BDGMPDEFAULT	Sets the Page Setup settings as the default settings.	"0" = Off "1" = On
DIAGFLD_BORDEROFFSET	Sets the Offset value in the Page Setup window. This field sets the border off from the report text by the specified value.	String Numeric value in hundredths of an inch.
DIAGFLD_BPRESENTATIONMENU	Toggles the automatic inclusion of the Presentation menu in the drawing toolbox. The extra symbols included in the Presentation menu include a basic shape for a computer, phone, person, disk and printer. The symbols are drawn on the diagram by the same method	"0" = Off "1" = On

DGMFLD constant	Description	Data Type
	as any other block symbol and may be named as appropriate.	
DIAGFLD_BREADONLY	Makes the diagram object read-only.	“0” = False “1” = True
DIAGFLD_BSHOWGRID	Toggles the automatic display of the underlying grid.	“0” = Off “1” = On
DIAGFLD_BSHOWLINESHADOW	Toggles the automatic inclusion of a shadow around all line symbols.	“0” = Off “1” = On
DIAGFLD_BSHOWNODESHADOW	Toggles the automatic inclusion of a shadow around all node symbols.	“0” = Off “1” = On
DIAGFLD_BSHOWPAGES	Toggles the Pages check box in the Diagram Display Options window. When checked the page print areas will be displayed as dotted lines so that if the diagram is printed in Actual Size mode then the page boundaries may be previewed.	“O” = unchecked “1” = checked
DIAGFLD_BSHOWRULER	Toggles the display of the x and y axis ruler markers measured in centimeters or inches depending upon the local settings on the PC. The x and y rules are saved with the drawing when toggled on.	“0” = Off “1” = On
DIAGFLD_BSHOWSCROLL	Toggles the automatic display of the scroll bars that allow the user to move around a diagram if it is larger that one	“O” = unchecked “1” = checked

DGMFLD constant	Description	Data Type
	screen in the current view mode. This option defaults to checked.	
DIAGFLD_BSHOWTEXTSHADOW	Toggles the automatic inclusion of a shadow around all text symbols.	“0” = Off “1” = On
DIAGFLD_BSNAPGRIDENT	Snaps all node symbols to the nearest grid line (normally invisible) on the diagram, after you have changed the grid setting to a coarser or finer grid setting.	“0” = Off “1” = On
DIAGFLD_BSNAPGRIDLIN	Snaps all line symbols to the nearest grid line (normally invisible) on the diagram, after you have changed the grid setting to a coarser or finer grid setting.	“0” = Off “1” = On
DIAGFLD_CGRAPHNAME	The name of the diagram.	String
DIAGFLD_CLEVELNUMBER	Level number of the diagram	Read-only String
DIAGFLD_DDDIAGRAM_DDIDENTITY	Data dictionary ID number for the diagram	Read-only Numeric
DIAGFLD_IDGMFORM	Selects the border form from the diagram border drop-down list in the Page Setup window.	“0” = no form/ no border “1” = IDEF0 Working “2” = IDEF0 Publication “3” = IDEF3 “4” = IDEF3 Released “5” = SSADM Form

DGMFLD constant	Description	Data Type
DIAGFLD_IGRAPHTYPE	The SA Type of diagram	String Internal constant number of diagram. A complete listing can be found in the diagrm.bas file in the SA directory.
DIAGFLD_PGRIDNUMENT	Node symbol Grid Points per inch.	"[Vertical] [Horizontal]" **Note:** must be used in conjunction with DIAGFLD_PGRIDSIZEENT
DIAGFLD_PGRIDNUMLIN	Line symbol Grid Points per inch.	"[Vertical] [Horizontal]" **Note:** must be used in conjunction with DIAGFLD_PGRIDSIZELIN
DIAGFLD_PGRIDSIZEENT	Node symbol Inches per Grid Point	"[Vertical] [Horizontal]" in hundredths of an inch. **Note:** must be used in conjunction with DIAGFLD_PGRIDNUMENT
DIAGFLD_PGRIDSIZELIN	Line symbol Inches per Grid Point	"[Vertical] [Horizontal]" in hundredths of an inch. **Note:** must be used in conjunction with DIAGFLD_PGRIDNUMLIN
DIAGFLD_PGRIDUNIT100	Sets the distance an object can be dragged within the grid. Default value="100 100"	"[Vertical] [Horizontal]" in hundredths of an inch.
DIAGFLD_PSHADOWDELTA	Sets the distance the shadow is placed from the symbol. Default value = "20 10"	"[Vertical] [Horizontal]" in hundredths of an inch.

DGMFLD constant	Description	Data Type
DIAGFLD_RGBSHADOWCOLOR	Sets shadow color.	“[RGB Color]”
DIAGFLD_RMARGIN	Sets Margins in Page Setup window.	“[Left] [Top] [Right] [Bottom]” in hundredths of an inch.
DIAGFLD_SAAUDITID	Diagram Audit ID	Read-only String
DIAGFLD_SAIDENTITY	Data dictionary ID number for the diagram.	Read-only Numeric
DIAGFLD_SALOCK	Locks the diagram.	“0” = Unlocked “1” = Locked
DIAGFLD_SAMAJORTYPE	Major Type (i.e. Diagram).	Read-only String
DIAGFLD_SAMAJORTYPENUMBER	Major Type Number (i.e. 1).	Read-only Numeric
DIAGFLD_SANAME	Name of diagram.	Read-only String
DIAGFLD_SANUMBER	Level Number of diagram (IDEF0 only)	Read-only Numeric
DIAGFLD_SATYPE	Type of diagram (e.g. Process Chart, Entity Relation, etc.).	Read-only String
DIAGFLD_SATYPENUMBER	Internal constant number of diagram.	Read-only Numeric
DIAGFLD_SAUPDATEDATE	Date of last update.	Read-only

DGMFLD constant	Description	Data Type
		Date Field
DIAGFLD_SAUPDATETIME	Time of last update.	Read-only Time Field
DIAGFLD_USEDENTCOUNT	Number of symbols on the diagram.	Long (Hexadecimal) Read-only
DIAGFLD_WBORDERPENSTYLE	Pen style of border in the Page Setup window.	"[SymPenStyle number]" For a complete listing of all SA Pen styles, see chapter 7.
DIAGFLD_WORIENTATION	Diagram Printing Orientation in the Page Setup window.	"0" = Printer Default "1" = Portrait "2" = Landscape "3" = Best Fit

Diagram Metrics

In the past, metrics have been used to create lists, run rules checks, and provide calculations for various Rational System Architect reports. Now it is possible for the user to run individual metrics by invoking the GetMetric method in the Diagram class. There exists a directory of all the diagram metrics in the DIAGRAMMETRIC enumerated list in the SA Object Browser. The following is a table of all the available diagram metrics and their descriptions.

Diagram Metric	Description
DIAGMETBALANCE	Compares the input and output lines of the diagram with the input and output lines of its parent process. Creates a list of non-matching input or output lines, giving the name and symbol type. (See Balance Parent Help file)
DIAGMETCHARCOUNT	Returns the number of characters in the description property of the diagram.
DIAGMETCURRENT	T/F Boolean Field. Returns True if diagram is currently displayed.
DIAGMETELELMENTLIST	Creates a list of the bottom level elements for all symbol definitions on the diagram. An element is bottom if it has no expanding relationships.
DIAGMETINPUTLIST	Creates a list of bottom level elements used as input for all symbol definitions on the diagram. An element is bottom if it has no expanding relationships.
DIAGMETLEVELNUMBER	Returns the number describing the hierarchal position of the diagram (i.e. 5.3, 5.3.1, etc.) as a string.
DIAGMETLEVELNUMBERSORT	Returns the number describing the hierarchal position of the diagram as a string. Each number contains 3 digits (i.e. 003.005.002). This allows the user to better sort the results.
DIAGMETLINECOUNT	Returns the number of lines in the description

Diagram Metric	Description
	property of a diagram.
DIAGMETOUTPUTLIST	Creates a list of bottom level elements used as output for all symbol definitions on the diagram. An element is bottom if it has no expanding relationships.
DIAGMETREFERENCE	Returns True if the diagram is referenced by any other object.
DIAGMETRULES	Runs rules check that looks for violations of standard methodology rules for the diagram.
DIAGMETSELECTED	Returns True if the diagram is currently opened and has any selected symbols.
DIAGMETTOP	Returns False if the diagram is the child of any symbols. Returns True if the diagram does not expand from a symbol.
DIAGMETUNMARKEDLIST	Creates a list of bottom level elements used by the definition of all line symbols on the diagram that are not marked as either input or output (no arrow heads). An element is bottom if it has no expanding relationships.
DIAGMETWORDCOUNT	Returns the number of words in the description property of the diagram.

7

The Symbol Class

Introduction

This is the Symbol Class with its attributes and methods pictured on the right.

Attributes

ArrowAtEnd

Purpose
This sets the associative properties for a line symbol. It creates an arrowhead at the terminating end of a line.

Parameters
Data Type: Boolean

Example
```
oSymbol.ArrowAtEnd = True
```

ArrowAtStart

Purpose
This sets the associative properties for a line symbol. It creates an arrowhead at the starting end of a line.

Parameters
Data Type: Boolean

Example
```
oSymbol.ArrowAtStart = True
```

AuditId

Purpose
All items symbols stored in Rational System Architect are tagged with the identity of the person who created or last modified the symbol, the identity is tagged to the symbol as an AuditId.

Parameters
Data Type: String

Read-only

ddId

Purpose
All symbols stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier. This method will return the identity of a symbol.

Parameters
Data Type: Long

Read-only

Definition

Purpose
Facilitates access to the definition class of the symbol.

Parameters
Read-only

Diagram

Purpose
Facilitates access to the diagram class on which the symbol has been created.

Parameters
Read-only

Encyclopedia

Purpose
Facilitates access to the encyclopedia class, to which the symbol belongs

Parameters
Read-only

FillColor

Purpose
The fill color for a symbol can be returned with this property. The value for the color is an OLE_COLOR value.

Parameters
An OLE_COLOR value is a BGR (Blue, Green, Red) value. To determine a BGR value, specify blue, green and red (each of which has a value from 0 - 255) in the following formula:

BGR value = (blue * 65536) + (green * 256) + red

FontColor

Purpose
The fill color for a symbol font can be returned with this property. The value for the color is an OLE_COLOR value.

Parameters
An OLE_COLOR value is a BGR (Blue, Green, Red) value. To determine a BGR value, specify blue, green and red (each of which has a value from 0 - 255) in the following formula:

BGR value = (blue * 65536) + (green * 256) + red

FromCardinality

Purpose
For relationship lines in an Entity Relation diagram or constraints on a Physical Data Model the cardinality at the ‘from end’ can be determined and set using this property.

The constants that are set or returned are as follows:

Constant	Number	Meaning
CARDINALITYZERO	0	Zero
CARDINALITYONLYONE	1	Only One
CARDINALITYZEROONE	2	Zero or One
CARDINALITYONEMULT	3	One or Many
CARDINALITYZEROONEMULT	4	Zero, One or Multiple
CARDINALITYMULT	5	Many
CARDINALITYUNKNOWN	6	Not Marked
CARDINALITYNOTUSED	7	No Cardinality

Handle

Purpose
This is the memory handle of the symbol only available at run-time. This handle is not unique and is rarely the same whenever accessed.

Parameters
Data Type: Long

Read-only

Example
```
Dim Handle As Long

Handle = oSymbol.Handle
```

LineStyle

Purpose
Lines drawn on diagrams can have one of many styles, which can be determined and set.

A few of the common constants that are set or returned are as follows:

Constant	Number	Meaning
LSARC	4	Elliptical Arc
LSAUTOSTROR	19	Automatic Straight Orthogonal
LSTRAA	1	Straight Any Orientation
LSTROR	3	Straight Orthogonal (Not Automatic)

MetaItem

Purpose
Facilitates access to the MetaItem's attributes.

Parameters
Read-only

Name

Purpose
The Symbol Name.

Parameters
Data Type: String

Read-only

PenColor

Purpose

This property returns or sets the color of the symbol pen. The value for the color is an OLE_COLOR value.

Parameters

An OLE_COLOR value is a BGR (Blue, Green, Red) value. To determine a BGR value, specify blue, green and red (each of which has a value from 0 - 255) in the following formula:

BGR value = (blue * 65536) + (green * 256) + red

PenStyle

Purpose

The style of the symbol pen can be returned or set with this property.

A range of constants prefixed with PEN exists. These correspond to the Format… Symbol Style… Pen option in Rational System Architect.

Constant	Number	Meaning
PENDASH	1	
PENDASH2DOT	4	
PENDASHDOT	3	
PENDOT	2	
PENNULL	5	No Pen Style
PENSOLID1	16	
PENSOLID2	48	
PENSOLID3	64	
PENSOLID4	128	
PENSOLID4A	384	

SAClass

Purpose
The class type of the symbol. Also known as the Major Type Number

Parameters
Data Type: Long

Read-only

Note: this will always be “2” for symbols.

SAType

Purpose
The numerical constant of the symbol.

Parameters
Data Type: Long

Read-only

Selected

Purpose
Whether or not the symbol is highlighted on the diagram.

Parameters
Data Type: Boolean

ToCardinality

Purpose

For relationship lines in an Entity Relation diagram or constraints on a Physical Data Model the cardinality at the to end can be determined and set using this property.

The constants that are set or returned are as follows:

Constant	Number	Meaning
CARDINALITYMULT	0	Many
CARDINALITYNOTUSED	1	No Cardinality
CARDINALITYONEMULT	2	One or Many
CARDINALITYONLYONE	3	Only One
CARDINALITYUNKNOWN	4	Not Marked
CARDINALITYZERO	5	Zero
CARDINALITYZEROONE	6	Zero or One
CARDINALITYZEROONEMULT	7	Zero, One or Multiple

TunnelAtEnd

Purpose
For line symbols in IDEF0 function diagrams “Tunneling” at the end of an arrow can be determined and set with this property.

Parameters
Data Type: Boolean

TunnelAtStart

Purpose
For line symbols in IDEF0 function diagrams “Tunneling” at the start of an arrow can be determined and set with this property.

Parameters
Data Type: Boolean

TypeName

Purpose
The type name of the symbol as a string, e.g. “Entity”

Parameters
Data Type: String

Read-only

UpdateDate

Purpose
The date the symbol was last modified.

Parameters
Data Type: String

Read-only

UpdateTime

Purpose
The time the symbol was last modified.

Parameters
Data Type: String

Read-only

Xpos

Purpose
The horizontal (X coordinate) of the symbol in 1/100ths of an inch.

This is for the location at the bottom right hand corner of the symbol and is taken from the left of the diagram.

Parameters
Data Type: Long

Xsize

Purpose
The width (X - axis) of the symbol. The unit measurement is 1/100ths of an inch.

Parameters
Data Type: Long

Ypos

Purpose
The vertical (Y coordinate) of the symbol in 1/100ths of an inch.

This is for the location at the bottom right hand corner of the symbol and is taken from the top of the diagram.

Parameters
Data Type: Long

Ysize

Purpose
The height (Y - axis) of the symbol. The unit measurement is 1/100ths of an inch.

Parameters
Data Type: Long

Methods

ConnectFrom

Purpose
Used to connect a symbol at the “from” end of the line.

Syntax
```
Symbol Object.ConnectFrom Line
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class from which the line symbol will be connected.

`Line`

Use: Required

Data Type: Symbol

Any instantiated line symbol

Example
```
Dim oFirstsymbol As Symbol, oSecondsymbol As Symbol,
  oLine As Symbol

Set oFirstsymbol = oDiagram.CreateSymbol("Customer",
  ETPROCESS)

Set oSecondsymbol = oDiagram.CreateSymbol("Order",
  ETPROCESS)

Set oLine = oDiagram.CreateSymbol("places",
  ETDATAFLOW)

Call oFirstsymbol.ConnectTo oLine

Call oSecondsymbol.ConnectFrom oLine
```

ConnectTo

Purpose
Used to connect a symbol at the “to” end of the line.

Syntax
```
Symbol Object.ConnectTo Line
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class to which the line symbol will be connected.

`Line`

Use: Required

Data Type: Symbol

Any instantiated line symbol

Example
```
Dim oFirstsymbol As Symbol, oSecondsymbol As Symbol,
  oLine As Symbol
Set oFirstsymbol = oDiagram.CreateSymbol("Customer",
  ETPROCESS)
Set oSecondsymbol = oDiagram.CreateSymbol("Order",
  ETPROCESS)
Set oLine = oDiagram.CreateSymbol("places",
  ETDATAFLOW)
Call oFirstsymbol.ConnectTo oLine
Call oSecondsymbol.ConnectFrom oLine
```

Delete

Purpose

Deletes the symbol from the diagram. When a diagram is closed, all symbols on it are automatically deleted unless the Diagram object is saved.

GetChildDiagrams

Purpose

This method retrieves an SAObjects collection of diagrams for a given symbol where a symbol has children attached.

The SAObjects collection will not be fully populated until the Complete flag for the collection is true. GetAllSymbols should be used in conjunction with either ReadAll or IsMoreThan methods.

Example

```
Dim oSymbol as Symbol, oCollectionofSymbols As
  SAObjects
Set oCollectionofSymbols = oSymbol.GetChildDiagrams
Call oCollectionofSymbols.ReadAll
```

GetField

Purpose

These are characteristics of the symbol, such as “Font Type”, “Font Height” of which some can be set such as “Font Type” and some that cannot such as “AuditId”.

Syntax

```
Symbol Object. GetField FieldID
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`FieldID`

Use: Required

Data Type: SYMBOLFIELDS

Symbol Field, see below for a complete list of all symbol fields.

Get Metric

Purpose
Calls certain lists, calculations, and pieces of internal functionality pertaining to symbols.

Syntax
```
Symbol Object.GetMetric Metric[, FieldType[, NbrChars[,
  NbrDec]]]
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`Metric`

Use: Required

Data Type: SYMBOLMETRIC

Symbol Metric, see below for a complete list of all symbol metrics.

`FieldType`

Use: Optional

Data Type: FLDTYPE

Field Type, see Chapter 17 for a complete list of Rational System Architect field types.

`NbrChars`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many characters are to be returned before the decimal point.

`NbrDec`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many numbers are to be returned after the decimal point.

GetProperty

Purpose

This returns the symbols property content for a given symbol property. Examine usrprops.txt and saprops.cfg for a complete list of all property names.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Junction of a IDEF3 Process Flow/ OV-6a diagram, the "Logic" property is actually a rename – the real property is “Junction Logic”. You would only know this if you looked up the definition of a Junction in saprops.cfg and saw that the property is actually called “Junction Logic” but has been ‘labeled’ “Logic”:

PROPERTY "Junction Logic" { EDIT Text ListOnly LIST "Junction Logic" LENGTH 3 DEFAULT "And" LABEL "Logic" }

Syntax

```
Symbol Object.GetProperty Name
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

GetPropertyAsCollection

Purpose

Some properties define relationships with other properties. For example an Entity may refer to its Model through its “Model” property. This method returns a collection of OneOf and ListOf diagrams or definitions. See Chapter 14 for more information on OneOf and ListOf property types.

Parameters

Data Type: OfCollection

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Junction of a IDEF3 Process Flow/ OV-6a diagram, the "Logic" property is actually a rename – the real property is “Junction Logic”. You would only know this if you looked up the definition of a Junction in saprops.cfg and saw that the property is actually called “Junction Logic” but has been ‘labeled’ “Logic”:

PROPERTY "Junction Logic" { EDIT Text ListOnly LIST "Junction Logic" LENGTH 3 DEFAULT "And" LABEL "Logic" }

Syntax

```
Symbol Object.GetPropertyAsCollection(PropName)
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`PropName`

Use: Required

Data Type: String

The name of the property as seen in the saprops.cfg

Example

```
Dim Models As OfCollection

    Set Models =
  SASym.GetPropertyAsCollection("Model")
```

GetRelatedObjects

Purpose
This method returns an SAObjects collection of related objects to the current symbol object.

Syntax
```
Symbol Object.GetRelatedObjects(RelType)
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`RelType`

Use: Required

Data Type: RELATETYPE

SA Relationship, see Chapter 16 for a complete list of all relationships.

Example
```
Dim oCollectionOfRelatedItems As SAObjects
Set oCollectionOfRelatedItems =
  oSymbol.GetRelatedObjects(RELCONNEND)
oCollectionOfRelatedItems.ReadAll
```

Save

Purpose
To save the symbol on a diagram after creation invoke the **save** method.

Example
```
oSymbol.Save
```

SetField

Purpose
This sets field values for a symbol, and requires two arguments, the field and its value.

Syntax
```
Symbol Object. SetField FieldID, value
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`FieldID`

Use: Required

Data Type: SYMBOLFIELD

Symbol Field, see below for a complete list of all Symbol fields.

`value`

Use: Required

Data Type: String

Value of symbol field

SetProperty

Purpose
By knowing the property name and its value, you can set a property of a symbol. Examine usrprops.txt and saprops.cfg for a complete list of all property names.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Junction of a IDEF3 Process Flow/ OV-6a diagram, the "Logic" property is actually a rename – the real property is “Junction Logic”. You would only know this if you looked up the definition of a Junction in

saprops.cfg and saw that the property is actually called “Junction Logic” but has been ‘labeled’ “Logic”:

PROPERTY "Junction Logic" { EDIT Text ListOnly LIST "Junction Logic" LENGTH 3 DEFAULT "And" LABEL "Logic" }

Syntax

```
Symbol Object.SetProperty Name, value
```

`Symbol Object`

Use: Required

Data Type: Object

Any instantiated Symbol class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

`Value`

Use: Required

Data Type: String

Value of symbol property

Symbol Fields

The symbol field property can contain a number of properties held about the symbol. Typically, contains information that a user cannot enter directly but is derived during normal use. Examples would include Update Time, Audit, Position, Size, TypeName.

Within the Object Model there is an enumerated type called *SYMBOLFIELD*. This is passed as a parameter to the operation **GetField(FieldID as *SYMBOLFIELD*)** and **SetField(FieldID as *SYMBOLFIELD*, Value as String)**. This enables the VBA programmer to both read and update low level symbol fields.

SYMBOLFIELD constant	Description	Data Type	
SYMFLD_AUDITID	Audit Id of the Symbol	String Read-only	
SYMFLD_BARRANGMENT	Arrangement of symbol tree.	"0" = Arrange Children Horizontally "1" = Arrange Children Vertically "2" = Arrange Children as a Block	
SYMFLD_BOTHERSYMBOLOGY	Displays alternate form of symbol (e.g. symbol sterotype)	Boolean	
SYMFLD_CBGCOLOR	Background color of a symbol on a Character Screen diagram.	"0" = Black "1" = Blue "2" = Green "3" = Cyan	"4" = Red "5" = Magenta "6" = Brown/yellow "7" = white
SYMFLD_CFGCOLOR	Foreground color of a symbol on a Character Screen	"0" = Black "1" = Blue	"4" = Red "5" = Magenta

SYMBOLFIELD constant	Description	Data Type	
	diagram.	“2” = Green “3” = Cyan	“6” = Brown/yellow “7” = white
SYMFLD_COCCOFFSET	Determines the spacing between each iteration of values for some input field on a Character Screen diagram.	Numeric	
SYMFLD_COCCURS	Allows multiple iteration of values for some input field on a Character Screen diagram.	Numeric	
SYMFLD_COMMENT	Sets the graphic comment property of the symbol.	String	
SYMFLD_CPROMPT	Cobalt Prompt Character of symbol on Character Screen diagram	1-bit String	
SYMFLD_CUNCLECOUNT	Returns the number of symbols that are directly attached to the parent symbol by flow lines.	Hexadecimal Read-only	
SYMFLD_DDCOMMENT	Data dictionary ID number of symbol’s comment symbol.	Numeric Read-only	
SYMFLD_DDIDENTITY	Data dictionary Id number of the symbol.	Numeric Read-only	
SYMFLD_DESCLOC	Location of the graphic comment of the symbol.	“XPos YPos” Numeric	
SYMFLD_DESCSIZE	Size of the graphic comment of the symbol.	“XSize YSize” Numeric	

SYMBOLFIELD constant	Description	Data Type
SYMFLD_DWSTYLE	Reflects what the user has selected as options for a graphic screen symbol on a Graphic Screen diagram.	Hexadecimal
SYMFLD_ENDLOC	Location of where line symbol ends.	“XPos YPos” Numeric
SYMFLD_ERROR1	First error detected by Rational System Architect.	Error number
SYMFLD_ERROR2	Second error detected by Rational System Architect.	Error number
SYMFLD_FONTFLAGS	Toggles Bold, Italic, Underline, or Strikethrough for Symbol font.	Hexadecimal “0x0002” = Bold “0x0005” = Italic “0x000B” = Underline “0x0010” = Strikethrough
SYMFLD_FONTHEIGHT	Symbol font size	Hexadecimal
SYMFLD_FONTNAME	Symbol font name (e.g. Arial, Times New Roman, etc.)	String
SYMFLD_FREXARCCHAR	Inserts an exclusive arc at the from end of a line symbol.	Boolean
SYMFLD_FROMCARDINALITY	Returns FromCardinality name (e.g. exactly one, one or many, etc.) For a complete list of FromCardinality values see FromCardinality attribute in Symbol class attributes.	String Read-only
SYMFLD_FROMCARDNUMB	Returns FromCardinality constant number. For a	Numeric

SYMBOLFIELD constant	Description	Data Type
ER	complete list of FromCardinality values see FromCardinality attribute in Symbol class attributes.	Read-only
SYMFLD_FROMCONNECTCOMPASSPOINT	Returns N,E,S, or W for start point of an ICOM Arrow connected at the “from” end	String Read-only
SYMFLD_HASFROMARROW	Returns true if line has an arrow at the “from” end	Boolean Read-only
SYMFLD_HASFROMTUNNEL	Returns true if ICOM arrow is tunneled at the “from” end	Boolean Read-only
SYMFLD_HASTOARROW	Returns true if line has an arrow at the “to” end	Boolean Read-only
Returns true if ICOM arrow is tunneled at the “to” end	Returns true if ICOM arrow is tunneled at the “to” end	Boolean Read-only
SYMFLD_LINESTYLE	Symbol line style	4-byte hexadecimal
SYMFLD_LOC	Location of symbol on the diagram.	“XPos YPos” Numeric
SYMFLD_NAME	Symbol name	String
SYMFLD_NAMECRLF	Name of Symbol with Carriage Return and Line Feed. In the name field, it is possible for the user to enter up to five lines of text.	String
SYMFLD_NAMECRLF1	If the name were to appear as string of continuous text (e.g JimJaneTomLouRon), the character number at	Numeric

SYMBOLFIELD constant	Description	Data Type
	which the text starts on the second line (i.e. 4).	
SYMFLD_NAMECRLF2	If the name were to appear as string of continuous text (e.g JimJaneTomLouRon), the character number at which the text starts on the third line (i.e. 8).	Numeric
SYMFLD_NAMECRLF3	If the name were to appear as string of continuous text (e.g JimJaneTomLouRon), the character number at which the text starts on the fourth line (i.e. 11).	Numeric
SYMFLD_NAMECRLF4	If the name were to appear as string of continuous text (e.g JimJaneTomLouRon), the character number at which the text starts on the fifth line (i.e. 14).	Numeric
SYMFLD_NAMELOC	Location of Symbol Name Field.	“XPos YPos” Numeric
SYMFLD_NAMESIZE	Size of Symbol Name Field	“XSize YSize” Numeric
SYMFLD_ORDER	Ordering of Association End	“O” = Unordered “1” = Ordered “2” = Sorted
SYMFLD_PENSTYLE	Symbol Pen style and width	4-byte hexadecimal
SYMFLD_ROTATION	Rotates Flag symbol on Structure Chart	Numeric values of 0 to 31 rotate flag symbol clockwise,

SYMBOLFIELD constant	Description	Data Type
		for example: “0” = south “8” = west “16” = north “24” = east
SYMFLD_SAMAJORTYPE	Major Type (i.e. Symbol)	String Read-only
SYMFLD_SAMAJORTYPENUMBER	Major Type number (i.e. 2)	Numeric Read-only
SYMFLD_SEQNUM	Entity Number on entity symbols.	Numeric
SYMFLD_SIZE	Symbol size	“XSize YSize” Numeric
SYMFLD_STARTLOC	Location of where line symbol starts.	“XPos YPos” Numeric
SYMFLD_STYLEFLAGS	Enables Symbol colors	Hexadecimal “0x0001” = Pen color “0x0002” = Fill color “0x0004” = Font color
SYMFLD_SUPERSUB	Sets Super-sub relationship value of the symbol	“0” = neither “1” = Super “2” = Sub
SYMFLD_TEXTFLAGS	Text Properties of symbol	Hexadecimal

SYMBOLFIELD constant	Description	Data Type
SYMFLD_TOCARDINALITY	Returns ToCardinality name (e.g. exactly one, one or many, etc.) For a complete list of ToCardinality values see ToCardinality attribute in Symbol class attributes.	String Read-only
SYMFLD_TOCARDNUMBER	Returns ToCardinality constant number. For a complete list of ToCardinality values see ToCardinality attribute in Symbol class attributes.	Numeric Read-only
SYMFLD_TOCONNECTCOMPASSPOINT	Returns N,E,S, or W for start point of an ICOM Arrow connected at the “to” end	String Read-only
SYMFLD_TOEXARCCHAR	Inserts an exclusive arc at the to end of a line symbol.	Boolean
SYMFLD_TYPE	Symbol SA Type	Internal Constant number of SAType.
SYMFLD_TYPENAME	Symbol SA type name (e.g. Entity, ICOM Arrow, etc.)	String Read-only
SYMFLD_U_S1_WPICTYPE	Picture Type (added graphic file to diagram).	String Read-only
SYMFLD_U_S1_ZPPICFILE	Path name of file being used to display picture.	String Read-only
SYMFLD_UPDATEDATE	Date of last update	Date field Read-only
SYMFLD_UPDATEDATEINTL	Date of last update (International Format)	Date Field

SYMBOLFIELD constant	Description	Data Type
		Read-only
SYMFLD_UPDATETIMEINTL	Time of last update (International Format)	Time Field Read-only
SYMFLD_XPENTITY	Internal number given to symbol.	Numeric Read-only
SYMFLD_XPGROUP	Internal number given to parent of the symbol.	Numeric Read-only
SYMFLD-XPLINK	Internal number of symbol that the selected symbol is linked to (e.g. Referent linked to a Unit of Behavior in an IDEF3 Process Flow diagram).	Numeric Read-only
SYMFLD_XPSIBLING	Internal number of next consecutive sibling.	Numeric Read-only
SYMFLD_XPSUBORDINATE	Internal number given to first child symbol	Numeric Read-only
SYMFLD_ZPDESC	Sets the graphic comment of the symbol.	String
SYMFLD-ZPSSADMSTR	Unknown.	

Symbol Metrics

In the past, metrics have been used to create lists, run rules checks, and provide calculations for various Rational System Architect reports. Now it is possible for the user to run individual metrics by invoking the GetMetric method in the Symbol class. There exists a directory of all the symbol metrics in the SYMBOLMETRIC enumerated list in the SA Object Browser. The following is a table of all the available symbol metrics and their descriptions.

Symbol Metric	Description
SYMMETANNOTATION	Returns True if the symbol is an annotation symbol (Doc Block, Text Box, Rectangle, Page Connector).
SYMMETBALANCE	Compares the input and output lines of the symbol with the input and output lines of its child process. Creates a list of non-matching input or output lines, giving the name and symbol type. (See Balance Child Help file) If the symbol is a data store, AND connector, or XOR connector, then it compares the defined elements and structures of those symbols. Creates list of undefined elements in the symbol and indicates whether the incoming and outgoing data flows have defined elements. (See Balance Horizontal Help file)
SYMMETBALANCEMSPEC	Balances the symbol definition's Minispec. Used for processes on Data Flow diagrams and modules on Structure Chart diagrams. For additional information, reference the Rational System Architect Help file, using keyword "Minispec".
SYMMETBOTTOM	Bottom is a derived T/F Boolean field. The value is true for a symbol if it does not expand to a diagram.
SYMMETCHARCOUNT	Returns the number of characters in the description property of the symbol.
SYMMETCONNECTOR	Returns True if the symbol is a connector symbol (AND connector, XOR connector, or ICOM Arrow Join)
SYMMETCURRENT	T/F Boolean Field. Returns True if symbol on currently

Symbol Metric	Description
	displayed diagram.
SYMMETELEMENTLIST	Creates a list of the bottom level elements for the symbol definition. An element is bottom if it has no expanding relationships.
SYMMETEXPRESSION	Creates a list of erroneous expression syntax or undefined data elements or data structures used by the expression of the symbol definition.
SYMMETICOMDEST	Returns the ICOM arrow destination role (input, control, mechanism, or boundary) as a string.
SYMMETICOMSOURCE	Returns the ICOM arrow source role (call, output, or boundary) as a string.
SYMMETINPUTLIST	Creates a list of bottom level elements used as input for the symbol definition. An element is bottom if it has no expanding relationships.
SYMMETISFOREIGNKEY	Returns True if the symbol definition is a foreign key.
SYMMETKEYCOMPNBR	Returns the component number of the primary key of a symbol. A user can alternatively view the component number by expanding the attribute list of the symbol definition in the browser detail (e.g. @1, @2, etc.).
SYMMETLEVELNUMBER	Returns the number describing the hierarchal position of the symbol (i.e. 5.3, 5.3.1, etc.) as a string.
SYMMETLEVELNUMBERSORT	Returns the number describing the hierarchal position of the symbol as a string. Each number contains 3 digits (i.e. 003.005.002). This allows the user to better sort the results.
SYMMETLINECOUNT	Returns the number of lines in the description property of a symbol definition.
SYMMETNORMALIZE1	Runs a check to see if the symbol definition is in first normal form. An entity is in First Normal Form if it contains no repeating groups.

Symbol Metric	Description
SYMMETNORMALIZE23	Runs a check to see if the symbol definition is in second and third normal form. An entity is in Second Normal Form if it is in First Normal Form and each non-key attribute is full functionally dependent on the primary key. An entity is in Third Normal Form if it is in Second Normal Form and each non-key attribute is dependent on the primary key and only on the primary key.
SYMMETOUTPUTLIST	Creates a list of bottom level elements used as output for the symbol definition. An element is bottom if it has no expanding relationships.
SYMMETPARENTSLASHDATA	Returns the symbol definition's foreign key slash data, which contains information on where the attribute is keyed from. A FK's slash data can also be viewed by expanding the symbol definition's attribute list in the browser details. The slash data will appear in the form of the following example: Row_Number / FKFROM "Stock_Location.Row_Number(stores)" /
SYMMETREFERENCE	Returns True if the symbol definition is referenced by any other object.
SYMMETRULES	Runs rules check that looks for violations of standard methodology rules for the symbol.
SYMMETSELECTED	Returns True if the symbol on a currently opened diagram is highlighted.
SYMMETSEQINPARENTSLIST	Checks the parent object's property set for list of child objects. Returns the number that the symbol definition appears in that list.
SYMMETTOP	Returns False if the diagram, which the selected symbol is on, is the child of any other symbols. Returns True if that diagram does not expand from any other symbol.
SYMMETUNMARKEDLIST	Creates a list of bottom level elements used by the definition of all line symbols that are not marked as either input or output (no arrow heads). An element is

Symbol Metric	Description
	bottom if it has no expanding relationships.
SYMMETUPDATEUSES	Updates the relation table of the data dictionary (RELATN.DBF). Has no return value.
SYMMETWORDCOUNT	Returns the number of words in the description property of the symbol definition.

8

The Definition Class

Topics in this chapter	Page

Introduction

This is the Definition Class with its attributes and methods pictured below.

Attributes

AuditID

Purpose
All items definitions stored in Rational System Architect are tagged with the identity of the person who created or last modified the definition; the identity is tagged to the definition as an AuditId.

Parameters
Data Type: String

Read-only

CheckedOut

Purpose
When set to True, the definition becomes read-only to everyone but the AuditID that checked it out.

Parameters
Data Type: Boolean

ddID

Purpose
All definition stored in Rational System Architect are uniquely identified internally by the use of a data dictionary identifier.

Parameters
Data Type: Long

Read-only

Encyclopedia

Purpose
Facilitates access to the parent encyclopedia class' attributes and methods.

Parameters
Read-only

Frozen

Purpose
User must have Freezing privileges in order to set this attribute. When set to True, the definition becomes read-only to everyone including the AuditID that froze the definition.

Parameter
Data Type: Boolean

Handle

Purpose
This is the memory handle of the definition only available at run-time. This handle is not unique and is rarely the same whenever accessed.

Parameters
Data Type: Long

Read-only

Example
```
Dim Handle As Long

Handle = oDefinition.Handle
```

Locked

Purpose
Returns a “True” or “False” value to indicate whether a definition is locked or not locked, i.e. in use by a user. This requires a valid definition object.

Parameters
Data Type: Boolean

Read-only

MetaItem

Purpose
Facilitates access to the MetaItem’s attributes.

Parameters
Read-only

Name

Purpose
The name of the specified definition object.

Parameters
Data Type: String

Read-only

ReadOnly

Purpose
Whether or not the definition is read-only

Parameters

Data Type: Boolean

Read-only

SAClass

Purpose
The class type of the definition. Also known as the Major Type Number

Parameters
Data Type: Long

Read-only

Note: this will always return “3” for a definition.

SAType

Purpose
The constant integer of the definition. All definitions in Rational System Architect have a unique numerical constant identifier.

Parameters
Data Type: Long

Read-only

TypeName

Purpose
The type of the definition as a string, e.g. Process

Parameters
Data Type: String

Read-only

UpdateDate

Purpose
The date the definition was last modified.

Parameters
Data Type: String

Read-only

UpdateTime

Purpose
The time the definition was last modified.

Parameters
Data Type: String

Read-only

xml

Purpose
The XML string of the definition. Manipulated by the GetXML method.

Parameters
Data Type: String

Read-only

Methods

Delete

Purpose
Deletes a definition specified by a definition object.

Example
```
Call oDefinition.Delete
```

GetField

Purpose
These are characteristics of the definition, such as “Type Number”, “Undefined Flag”, “Major Type Name”.

Syntax
```
Definition Object. GetField FieldID
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`FieldID`

Use: Required

Data Type: DEFFLD

Definition Field, see below for a complete list of all Definition fields.

Get Metric

Purpose
Calls certain lists, calculations, and pieces of internal functionality pertaining to definitions.

Syntax
```
Definition Object.GetMetric Metric[, FieldType[,
  NbrChars[, NbrDec]]]
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`Metric`

Use: Required

Data Type: DEFINITIONMETRIC

Definition Metric, see below for a complete list of all Definition metrics.

`FieldType`

Use: Optional

Data Type: FLDTYPE

Field Type, see Chapter 17 for a complete list of Rational System Architect field types.

`NbrChars`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many characters are to be returned before the decimal point.

`NbrDec`

Use: Optional

Data Type: Long

If the Field Type has been entered, then this parameter tells SA how many numbers are to be returned after the decimal point.

GetProperty

Purpose
This returns the property content for any given definition property.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Service Time Profile, the " Time Units " property is actually a rename – the real property is "Duration Time Units". You would only know this if you looked up the definition of a Service Time Profile in saprops.cfg and saw that the property is actually called "Duration Time Units" but has been 'labeled' "Time Units":

PROPERTY "Duration Time Units" { EDIT Text LISTONLY List "Time Units" LABEL "Time Units" DEFAULT "Hour" LENGTH 20 READONLY }

Syntax

```
Definition Object.GetProperty Name
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

GetPropertyAsCollection

Purpose
Some properties define relationships with other properties. For example an Entity Definition refers to Attributes through its "Description" property. This method returns a collection of

OneOf and ListOf diagrams or definitions. See Chapter 14 for more information on OneOf and ListOf property types.

Parameters

Data Type: OfCollection

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Service Time Profile, the " Time Units " property is actually a rename – the real property is "Duration Time Units". You would only know this if you looked up the definition of a Service Time Profile in saprops.cfg and saw that the property is actually called "Duration Time Units" but has been 'labeled' "Time Units":

PROPERTY "Duration Time Units" { EDIT Text LISTONLY List "Time Units" LABEL "Time Units" DEFAULT "Hour" LENGTH 20 READONLY }

Syntax

```
Definition Object.GetPropertyAsCollection(PropName)
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`PropName`

Use: Required

Data Type: String

The name of the property as seen in the saprops.cfg

Example

```
Dim i As Long, DiagId As Long

i = 0

Do While sa.Encyclopedia.GetFilteredDefinitions("",
  DFXACTIVITY).IsMoreThan(i)

    i = i + 1

    Dim AttribColl As OfCollection

    Set SADef =
  sa.Encyclopedia.GetFilteredDefinitions("",
  DFXACTIVITY).Item(i)
```

```
        Set AttribColl =
      SADef.GetPropertyAsCollection("Description")
  Loop
```

GetRelatedObjects

Purpose
This method returns an SAObjects collection of related objects to the current definition object.

Syntax
```
Definition Object.GetRelatedObjects(RelType)
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`RelType`

Use: Required

Data Type: RELATETYPE

SA Relationship, see Chapter 16 for a complete list of all relationships.

GetXML

Purpose
Exports the definition's XML string into a valid .xml file name.

Syntax
```
Definition Object.GetXML(strXMLTextOut)
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`StrXMLTextOut`

Use: required

Data Type: String

A valid .xml file, to which SA will export the definition's XML string.

Save

Purpose
Used to save an instance of a definition.

Example
```
Call oDefinition.Save
```

SetField

Purpose
SetField allows a Definition Field to be set with a specified value.

Syntax
```
Definition Object.SetField FieldID, value
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`FieldID`

Use: Required

Data Type: DGMFLD

Definition Field, see below for a complete list of all Definition fields.

`Value`

Use: Required

Data Type: String

Value of Definition field

SetProperty

Purpose

The setting of a definition property value requires the name of the property as the first argument and the value to set as the second. The property names are found in the saprops.cfg and usrprops.txt files.

Parameters

Oftentimes the real name of a property is not the same as what shows up on a definition dialog; for example, in a Service Time Profile, the " Time Units " property is actually a rename – the real property is "Duration Time Units". You would only know this if you looked up the definition of a Service Time Profile in saprops.cfg and saw that the property is actually called "Duration Time Units" but has been 'labeled' "Time Units":

PROPERTY "Duration Time Units" { EDIT Text LISTONLY List "Time Units" LABEL "Time Units" DEFAULT "Hour" LENGTH 20 READONLY }

Syntax

```
Definition Object.SetProperty Name, value
```

`Definition Object`

Use: Required

Data Type: Object

Any instantiated Definition class

`Name`

Use: Required

Data Type: String

The Name of the property as seen in the saprops.cfg

`Value`

Use: Required

Data Type: String

Value of Definition property

Definition Fields

The definition field property can contain a number of properties held about the definition. They typically contain information that a user cannot enter directly but is derived during normal use. Within the Object Model there is an enumerated type called *DEFFLD*. This is passed as a parameter to the operation **GetField(FieldID as *DEFFLD*)** and **SetField(FieldID as *DEFFLD*, Value as String)**. This enables the VBA programmer to both read and update low level definition fields.

DEFFLD constant	Description	Data Type
DEFNFLD_SAAUDITID	Audit Id of the definition.	Read-only
DEFNFLD_SAIDENTITY	Data dictionary Id of the definition.	Read-only
DEFNFLD_SAIDENTITY4	Data dictionary Id of the definition.	4-byte binary number Read-only
DEFNFLD_SAISUNDEFINED	Returns “T” if definition is undefined, “F” if definition is defined.	Read-only
DEFNFLD_SALOCK	Locks the definition.	“T” = Locked “F” = Unlocked
DEFNFLD_SAMAJORTYPE	Major type (i.e. “Definition”)	Read-only
DEFNFLD_SAMAJORTYPENUMBER	Major type number (i.e. “3”)	Read-only
DEFNFLD_SANAME	Name of definition.	String
DEFNFLD_SATYPE	SA type of definition (e.g. UML Class, Entity, etc.)	Read-only
DEFNFLD_SATYPENUMBER	Internal constant number of the definition type. For a complete listing, reference the DEFNS.BAS file in the SA directory	Read-only

DEFFLD constant	Description	Data Type
DEFNFLD_SAUPDATEDATE	Date of last update.	Read-only
DEFNFLD_SAUPDATETIME	Time of last update.	Read-only

Definition Metrics

In the past, metrics have been used to create lists, run rules checks, and provide calculations for various Rational System Architect reports. Now it is possible for the user to run individual metrics by invoking the GetMetric method in the Definition class. There exists a directory of all the definition metrics in the DEFINITIONMETRIC enumerated list in the SA Object Browser. The following is a table of all the available definition metrics and their descriptions.

Definition Metric	Description
DEFMETBOTTOM	Bottom is a derived T/F Boolean field. The value is true for definitions that are not expressions, and expressions that do not contain any data elements or data structures.
DEFMETCURRENT	T/F Boolean Field. Returns True if the symbol of the definition is on a currently displayed diagram.
DEFMETEXPRESSION	Creates a list of erroneous expression syntax or undefined data elements or data structures used by the expression of the definition.
DEFMETISFOREIGNKEY	Returns True if the definition is a foreign key.
DEFMETKEYCOMPNBR	Returns the component number of the primary key of a definition. A user can alternatively view the component number by expanding the attribute list of the definition in the browser detail (e.g. @1, @2, etc.).
DEFMETNORMALIZE1	Runs a check to see if the definition is in first normal form. An entity is in First Normal Form if it contains no repeating groups.
DEFMETNORMALIZE23	Runs a check to see if the definition is in second and third normal form. An entity is in Second Normal Form if it is in First Normal Form and each non-key attribute is full functionally dependent on the primary key. An entity is in Third Normal Form if it is in Second Normal Form and each non-key attribute is

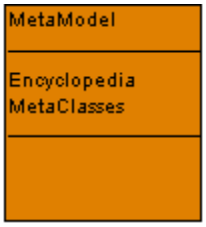
Definition Metric	Description
	dependent on the primary key and only on the primary key.
DEFMETPARENTSLASHDATA	Returns the definition's foreign key slash data, which contains information on where the attribute is keyed from. A FK's slash data can also be viewed by expanding the definition's attribute list in the browser details. The slash data will appear in the form of the following example: Row_Number / FKFROM "Stock_Location.Row_Number(stores)" /
DEFMETREFERENCE	Returns True if the definition is referenced by any other object.
DEFMETSELECTED	Returns True if there is a selected symbol defined by the definition.
DEFMETSEQINPARENTSLIST	Checks the parent object's property set for list of child objects. Returns the number that the definition appears in that list.
DEFMETSYNCHRONIZE	If the user has already specified that the definition should be synchronized with another object, then this metric executes the synchronization. Synchronization can be set in the SA2001.INI Editor
DEFMETUPDATEUSES	Updates the relation table of the data dictionary (RELATN.DBF). Has no return value.
DEFNMETCHARCOUNT	Returns the number of characters in the description property of the definition.
DEFNMETLINECOUNT	Returns the number of lines in the description property of a definition.
DEFNMETWORDCOUNT	Returns the number of words in the description property of the definition.

9

The MetaModel Class

Introduction

The MetaModel Object provides access to the metaclasses objects for an encyclopedia.



Attributes

Encyclopedia

Purpose
Refers to the parent encyclopedia object of the metamodel object.

Parameters
Read-only

MetaClasses

Purpose
Provides access to the collection of MetaClass Objects for an encyclopedia.

Parameters
Data Type: SA Collection

Read-only

Example

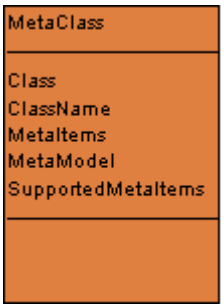
```
Dim coll As SACollection, i As Integer
Set coll = sa.Encyclopedia.metamodel.MetaClasses
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).Class
  Next i
```

10

The MetaClass Class

Introduction

The MetaClass object returns information about class level information for Objects in the repository.



Attributes

Class

Purpose
The class number of the object in the repository. The return values are 1 for a Diagram, 2 for a Symbol and 3 for a Definition. Also known as the Major Type Number.

Parameters
Data Type: Long

Read-only

Example

```
Dim coll As SACollection, i As Integer
Set coll = sa.Encyclopedia.metamodel.MetaClasses
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).Class
  Next i
```

ClassName

Purpose
The name of the class. The valid class names in an Encyclopedia are Diagrams, Symbols and Definitions. Also known as the Major Type.

Parameters
Data Type: String

Read-only

Example

```
Dim coll As SACollection, i As Integer
Set coll = sa.Encyclopedia.metamodel.MetaClasses
  For i = 1 To coll.Count
```

```
    Debug.Print coll.Item(i).ClassName
  Next i
```

MetaItems

Purpose
This property provides access to the MetaItems class for all metaitems.

Parameters
Data Type: SACollection

Read-only

Example
```
Dim coll As SACollection, i As Integer
Set coll = sa.Encyclopedia.metamodel.MetaClasses
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).MetaItems.Count
  Next i
```

MetaModel

Purpose
This provides access to the parent MetaModel class.

Parameters
Read-only

SupportedMetaItems

Purpose
This property provides access to only the metaitems that are turned on for this encyclopedia.

Parameters
Data Type: SACollection

Read-only

Example

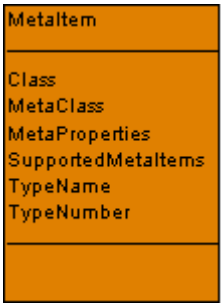
```
Dim sa As SA2001.Application

Set sa = New SA2001.Application

Debug.Print
  sa.Encyclopedia.MetaModel.MetaClasses.Item(1).Suppo
  rtedMetaItems.Count

Set sa = Nothing
```

11

The MetaItem Class

Introduction

This object provides information about an individual object type in the encyclopedia. It corresponds to information in saprops.cfg and usrprops.txt for specific types of object.



Attributes

Class

Purpose
The class of the object. The valid values are 1 for a Diagram, 2 for a symbol and 3 for a definition. Also known as the Major Type Number.

Parameters
Data Type: Long

Read-only

Example
```
Dim sa As SA2001.Application

Set sa = New SA2001.Application

With
  sa.Encyclopedia.MetaModel.MetaClasses.Item(1).Suppo
  rtedMetaItems.Item(1)

    Debug.Print .Class

End With
```

MetaClass

Purpose
This property provides access to the parent MetaClass object.

Parameters
Read-only

MetaProperties

Purpose

This property provides access to all of the properties supported by this object type as a collection.

Parameters
Data Type: SACollection

Read-only

Example

```
Dim coll As SACollection, i As Integer
Set coll = Definition.MetaItem.MetaProperties
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).Name
  Next i
```

SupportedMetaItems

Purpose
This property provides access to only the metaitems that are turned on for this encyclopedia.

Parameters
Data Type: SACollection

Read-only

TypeName

Purpose
The name of the type of object.

Parameters
Data Type: String

Read-only

Example

```
Dim sa As SA2001.Application
Set sa = New SA2001.Application
```

```
With
  sa.Encyclopedia.MetaModel.MetaClasses.Item(1).Suppo
  rtedMetaItems.Item(1)
    Debug.Print .TypeName
End With
```

TypeNumber

Purpose
The numerical constant assigned to this object type by Rational System Architect.

Parameters
Data Type: Long

Read-only

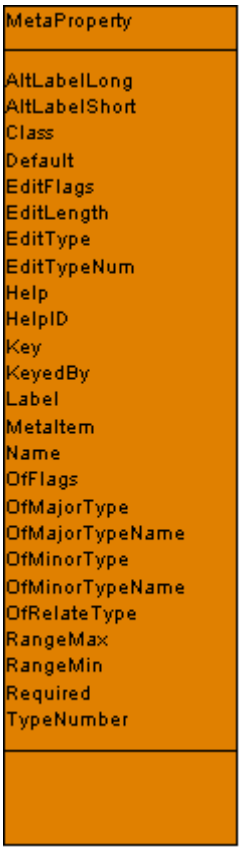
Example
```
Dim sa As SA2001.Application
Set sa = New SA2001.Application
With
  sa.Encyclopedia.MetaModel.MetaClasses.Item(1).Suppo
  rtedMetaItems.Item(1)
    Debug.Print .TypeNumber
End With
```

12

The MetaProperty Class

Introduction

The MetaProperty object allows you to retrieve information about each property of a specific object in the encyclopedia. This object corresponds to the Property keyword in saprops.cfg and usrprops.txt files.



Attributes

The following properties can all be retrieved using the following example and by replacing name with one of the described properties below.

```
Dim sa As SA2001.Application
Set sa = New SA2001.Application
With
  sa.Encyclopedia.MetaModel.MetaClasses.Item(1).Suppo
  rtedMetaItems.Item(1).MetaProperties.Item(1)
    Debug.Print .Name
End With
Set sa = Nothing
```

AltLabelLong
The Alternative Rational System Architect long label for the property.

Data Type: String

Read-only

AltLabelShort
The Alternative Rational System Architect short label for the property.

Data Type: String

Read-only

Class
The properties' parent object class type. Also known as the Major Type Number.

Data Type: Long

Read-only

Default
The default value set for the property on first use.

Data Type: String

Read-only

EditFlags
The number of properties that must be filled in for a property.

Data Type: Long

Read-only

EditLength
The edit length of the property.

Data Type: Long

Read-only

EditType
See table below for a list of SAEditTypes

Data Type: String

Read-only

EditTypeNum
See table below for a list of SAEditType numerical constants.

Data Type: Long

Read-only

Help
The help text set for this property.

Data Type: String

Read-only

HelpID
The ID number of the help text.

Data Type: Long

Read-only

Key
Whether the property is a key or not.

Data Type: Boolean

Read-only

KeyedBy
The property is keyed by another property or not.

Data Type: SACollection

Read-only

Label
The standard Rational System Architect label for the property.

Data Type: String

Read-only

MetaItem
Provides access to the parent MetaItem object.

Read-only

Name
The name of the property.

Data Type: String

Read-only

OfFlags
The number of flags for the property.

Data Type: Long

Read-only

OfMajorType
The numerical constant of the parent objects class.

Data Type: Long

Read-only

OfMajorTypeName
The type name of the parent objects class.

Data Type: String

Read-only

OfMinorType
The current parent objects numerical type constant.

Data Type: Long

Read-only

OfMinorTypeName
The current parent objects type name.

Data Type: String

Read-only

OfRelateType
The parent objects relationship number. See Chapter 15 for a complete list of all relationship numbers.

Data Type: Long

Read-only

RangeMax
The maximum edit range value.

Data Type: Long

Read-only

RangeMin
The minimum edit range value.

Data Type: Long

Read-only

Required
The required property for the object.

Data Type: Long

Read-only

TypeNumber
The type number of the parent object.

Data Type: Long

Read-only

EditType	Number	Description
Text	1	The property defined as text may be a list, or any alphanumeric characters typed in by the user.
Date	2	The property's length must be 10 characters and is based on the date format set in Windows.
Numeric	3	Specifies that the property must contain values that are numbers.
Boolean	4	The property has one of two values: True (T) or False (F); and will appear in a definition dialog as a check box.
Expression	5	The value of the property must be entered as a series of strings separated by the + sign. This word has been replaced by EXPRESSIONOF.
Minispec	6	A property whose value expresses the processing logic of a process symbol. Minispecs are written using a formal syntax often referred to as Structured English.
Time	7	The property will contain a time stamp, in the notation appropriate to the time format defined to Windows.
ListOf	8	Forms a one-to-one or many relationship between the current definition type and the definition type named after the ListOf setting for all items in the list property.
ExpressionOf	9	Forms a one-to-one or many relationship between the current definition type and the definition type named after the ExpressionOf setting for all items which exist in the property.
OneOf	10	Forms a one-to-one relationship between the current definition type and the definition type named after the OneOf setting for the item that exists in the property.
TemplateOf	11	Syntax of a trigger template.

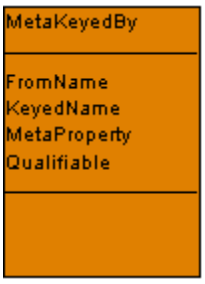
EditType	Number	Description
ParmListOf	12	Forms a one-to-one or many relationship between the current definition type and the definition type that is specified by its name and parameters and named after the ParmListOf setting for all items in the list property.
ParmOneOf	13	Forms a one-to-one relationship between the current definition type and the definition type that is specified by its name and parameters and named after the OneOf setting for the item that exists in the property.

13

The MetaKeyedBy Class

Introduction

This is the MetaKeyedBy class with its attributes pictured below.



Topics in this chapter	Page
Attributes	13-2

Attributes

FromName

Purpose
The From Name property for key qualification of a MetaProperty.

Parameters
Data Type: String

Read-only

KeyedName

Purpose
The Keyed Name of the MetaProperty.

Parameters
Data Type: String

Read-only

MetaProperty

Purpose
Provides access to the parent MetaProperty class.

Parameters
Read-only

Qualifiable

Purpose
Value of whether or not the MetaProperty is qualifiable, meaning it maintains reference to its key structure.

Parameters
Data Type: Boolean

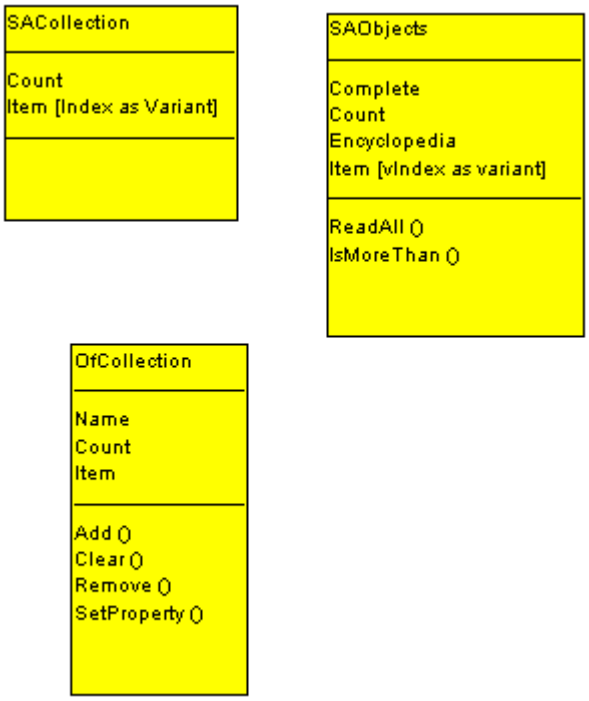
Read-only

14

Rational System Architect Collections

Introduction

Here is a picture of the Rational System Architect Collection class and their attributes.



SAObjects Class

This collection class is used to manipulate a group of similar Rational System Architect objects (diagram, symbols, or definitions).

Complete

Purpose
Boolean value indicating true when all possible members have been read into the collection.

Parameters
Data Type: Boolean

Read-only

Example
```
Dim coll As SAObjects, i As long
Set coll = sa.Encyclopedia.GetAllDiagrams
i = 1
  Do While coll.IsMoreThan(i)
     i = i +1
     Debug.Print coll.Item(i).Name
  Loop
Debug.Print coll.Complete
Debug.Print coll.Count
```

Count

Purpose
The value representing the number of collection members.

Parameters
Data Type: Long

Read-only

Example

```
Dim coll As SAObjects, i As Integer
Set coll = sa.Encyclopedia.GetAllDiagrams
coll.ReadAll
  For i = 1 To coll.Count
     Debug.Print coll.Item(i).Name
  Next i
```

Encyclopedia

Purpose
Reference back to the encyclopedia class of the current object in the collection.

Parameters
Read-only

Item(Index)

Purpose
Each member of the collection has a specific index number within the collection derived when the collection is created. The item variable is an object based upon the index of the collection. Its index number or name, if known, may reference this object.

Parameters
Data Type: Boolean

Read-only

Example

```
Dim coll As SAObjects, i As Integer
Set coll = sa.Encyclopedia.GetAllDiagrams
coll.ReadAll
  For i = 1 To coll.Count
```

```
    Debug.Print coll.Item(i).Name
  Next i
```

IsMoreThan(Index)

Purpose
Returns true if there are more items in the collection than the current value of the index – used when reading through a collection one by one.

Parameters
Data Type: Boolean

Example
```
Dim coll As SAObjects, i As long
Set coll = sa.Encyclopedia.GetAllDiagrams
i = 1
  Do While coll.IsMoreThan(i)
     i = i +1
  Loop
```

ReadAll

Purpose
Reads all possible occurrences of the type into the collection.

Example
```
Dim coll As SAObjects, i As Integer
Set coll = sa.Encyclopedia.GetAllDiagrams
coll.ReadAll
  For i = 1 To coll.Count
     Debug.Print coll.Item(i).Name
Next i
```

SACollection Class

This collection class is used to manipulate a group of similar Rational System Architect properties.

Count

Purpose
Returns a numeric value for the number of collection members.

Parameters
Data Type: Long

Read-only

Example
```
Dim coll As SACollection, i As Integer
Set coll = Definition.MetaItem.MetaProperties
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).Name
  Next i
```

Item(Index)

Purpose
Each member of the collection has a specific index number within the collection derived when the collection is created. The item variable is an object based upon the index of the collection. Its index number or name, if known, may reference this object.

Parameters
Data Type: Boolean

Read-only

Example
```
Dim coll As SACollection, i As Integer
```

```
Set coll = Definition.MetaItem.MetaProperties
  For i = 1 To coll.Count
    Debug.Print coll.Item(i).Name
  Next i
```

OfCollection Class

This collection class is used to manipulate a group of diagrams or definitions that are components of a special property type listed in the table below.

Property Type	Description
ListOf	Forms a one-to-one or many relationship between the current definition type and the definition type named after the ListOf setting for all items in the list property.
OneOf	Forms a one-to-one relationship between the current definition type and the definition type named after the OneOf setting for the item that exists in the property.

Name

Purpose
Name of the property list.

Parameters
Data Type: String

Read-only

Example
```
Dim oDef as Definition, coll as OfCollection
Set coll = oDef.GetPropertyAsCollection("Operations")
  Debug.Print coll.Name
```

Count

Purpose
A numeric value representing the number of collection members.

Parameters

Data Type: Long

Read-only

Example

```
Dim oDef as Definition, coll as OfCollection, i as integer
Set coll = oDef.GetPropertyAsCollection("Operations")
  Debug.Print coll.Name
  For i = 1 to coll.Count
     Debug.Print coll.Item(i).Name
  Next i
```

Item

Purpose

Each member of the collection has a specific index number within the collection derived when the collection is created. The item variable is an object based upon the index of the collection. Its index number or name, if known, may reference this object.

Parameters

Data Type: Boolean

Read-only

Example

```
Dim oDef as Definition, coll as OfCollection, i as integer
Set coll = oDef.GetPropertyAsCollection("Operations")
  Debug.Print coll.Name
  For i = 1 to coll.Count
     Debug.Print coll.Item(i).Name
  Next i
```

Add

Purpose
Adds a diagram or definition object to the property list

Syntax
```
OfCollection Object.Add(Item[, Before[, After]]
```

`OfCollection Object`

Use: Required

Data Type: Object

Any instantiated OfCollection class

`Item`

Use: Required

Data Type: String

Diagram or definition object to be added to the property list.

`Before`

Use: Optional (cannot be set if After parameter is already set)

Data Type: Long

Number of diagram or definition object in collection object count after which the new diagram or definition object will be added.

`After`

Use: Optional (cannot be set if Before parameter is already set)

Data Type: Long

Number of diagram or definition object in collection object count before which the new diagram or definition object will be added.

Example
```
Dim i As Integer, oDef As Definition, coll As OfCollection
Set coll = oDef.GetPropertyAsCollection("Location Types")
  coll.Add Chr(34) & "Regional Office" & Chr(34)
  coll.SetProperty
```

```
  oDef.Save
```

Clear

Purpose
Clears the OfCollection object of all items in the collection.

Example
```
Dim i As Integer, oDef As Definition, coll As OfCollection

Set coll = oDef.GetPropertyAsCollection("Location Types")

  coll.Clear

  oDef.Save
```

Remove

Purpose
Removes a diagram or definition object from the collection.

Syntax
```
OfCollection Object.Remove(Index)
```

`OfCollection Object`

Use: Required

Data Type: Object

Any instantiated OfCollection class

`Index`

Use: Required

Data Type: Long

The number of the diagram or definition item in the collection object count to remove.

Example

```
Dim i As Integer, oDef As Definition, coll As OfCollection
Set coll = oDef.GetPropertyAsCollection("Location Types")
  coll.Remove(2)
  coll.SetProperty
  oDef.Save
```

SetProperty

Purpose

Saves the property list within the OfCollection object. It is also important that the definition object itself is saved as this contains the property list to which the collection is saved.

Example

```
Dim i As Integer, oDef As Definition, coll As OfCollection
Set coll = oDef.GetPropertyAsCollection("Location Types")
  coll.Add Chr(34) & "Regional Office" & Chr(34)
  coll.SetProperty
  oDef.Save
```

15

Rational System Architect Events

Introduction

There may be occasions when control needs to be exercised over what action a user performs and the response of Rational System Architect to those actions. This may be to prohibit certain operations, provide warnings, and enforce behavior or to perform additional operations. On those occasions it is necessary to identify the action explicitly at the time of its occurrence and to respond accordingly.

Actions that occur at a given point of time are referred to as “Events” and these can be handled in Rational System Architect as and when they occur.

Rational System Architect recognizes a number of events such as when a symbol is placed on a diagram, a diagram is saved or an encyclopedia is opened. The following table lists all the events that are recognized, with details of variables that are passed.

Application Events

AuditIDChanged

Purpose
AuditID changes, new AuditId is passed as NewAuditID. The Audit ID can be changed from the **File, Audit ID…** menu.

Syntax
AuditIDChanged(NewAuditID)

NewAuditID

Use: Required

Data Type: String

User name that the AuditID property is changing to.

Example

DiagramClose

Purpose
Diagram is closed.

Syntax
DiagramClose(hdgm)

hdgm

Use: Required

Data Type: Long

Diagram handle is passed as hDgm.

Example

DiagramOpen

Purpose
Diagram is opened.

Syntax
DiagramOpen(hdgm)

hDgm

Use: Required

Data Type: Long

Diagram handle is passed as hDgm.

Example

DiagramSave

Purpose
Diagram is saved.

Example

EncyClose

Purpose
Encyclopedia is closed.

Example

EncyOpen

Purpose
Encyclopedia is opened.

Example

MainMenuUpdate

Purpose
Main menu is updated. This is the File menu and other associated menu entries.

Example

MethodMenuUpdated

Purpose
Draw menu is updated. This is when a new method type forces a menu redraw and addition of new menu items.

Example

ReportsMenuUpdate

Purpose
Reports menu is updated. This is whenever the reports menu is modified by Rational System Architect to reflect method specific reports or rules checks.

Example

ToolsMenuUpdate

Purpose
Tools menu is updated. This is whenever the tools menu is updated to reflect tools options applicable to certain methods being in use.

Example

ShowNode

Purpose

Event triggered when browser VBA filter is active. Set the RetCode to show/hide this item.

Syntax

```
ShowNode(TabIndex, TabName, ddId, Major, Minor, Name, Memo,
  LockFlags, RetCode)
```

`TabIndex`

Use: Required

Data Type: Integer

Tab Name	Tab Index number
All Methods	1
Data Modeling	2
Business Process	3
OO Legacy	4
Application	5
Structured	6
Organization	7
Technology	8
Location	9
Business Direction	10
UML	11
XML	12

`TabName`

Use: Required

Data Type: String

See table above

`ddId`

Use: Required

Data Type: Long

`Major`

Use: Required

Data Type: Long

`Minor`

Use: Required

Data Type: Long

`Name`

Use: Required

Data Type: String

`Memo`

Use: Required

Data Type: String

`LockFlags`

Use: Required

Data Type: Long

`RetCode`

Use: Required

Data Type: NodeStatus

Node Status	Number
StatusDONTSHOW	0
StatusSHOW	1
StatusDONTCARE	2

Example

ShutDown

Purpose
Rational System Architect is shutdown.

Example

StartUp

Purpose
Rational System Architect is started up.

Example

SymbolEvent

Purpose
Symbol is placed on a diagram.

Syntax
```
SymbolEvent(hDgm, hSym, hSymOther, SymEvent, lData)
hDgm
```

Use: Required

Data Type: Long

Handle of diagram

`hSym`

Use: Required

Data Type: Long

Handle of symbol

`hSymOther`

Use: Required

Data Type: Long

Handle of line symbol (to which the node symbol is being connected/disconnected).

`SymEvent`

Use: Required

Data Type: SYMEVENTS

SYMEVENTS	Number	Description
ADDCONN	64	Node symbol is connected with a line symbol
BREAKCONN	65	Node symbol's connection with a line symbol is deleted.
SYM_DESELECTED	257	Symbol is deselected.
SYM_SELECTED	256	Symbol is selected.
TRANSFORMED	66	Symbol is transformed into another symbol. This can be done on certain symbols by right-clicking on the selected symbol and choosing Transform.

`Ldata`

Use: Required

Data Type: Long

Guidelines for Adding Macro Items to Menus Programmatically

As of V10.1 Rational System Architect has almost completely released the menus and toolbars from their previous fixed format. This means that users are now able to customize menus, and their customizations will remain. The only exceptions to this are the Draw menu and toolbar, where customization is only possible after the default list of drawing tool commands or buttons.

The goal of this change is to enable tailored menu and toolbar configurations for specific user groups via SA Catalog Manager.

In Rational System Architect V10.0 and before, macro menu code tended to be written taking into account the fact that Rational System Architect would destroy certain menus, and force them to be redrawn. This destruction meant that users who required menu items to disappear during a session, could save themselves the need to actually remove an item from a menu programmatically. A typical example might be when a diagram was closed – the tools menu would be destroyed and re-created. Thus the user would only be required to re-insert the menu item if it was required at that time. Now that Rational System Architect does not destroy menus, these menu items will remain visible unless the macro code is modified to remove (hide) them. Instructions for modifying such macros built in Rational System Architect 10.0 and before are provided in the Rational System Architect's Conversion manual, called Conversion.pdf, located on the IBM support site.

Guidelines for adding macros to menus in Rational System Architect V10.1 and later are provided in this section, below.

How Do I Add a Macro to a Standard Rational System Architect Menu ?

The main SA2001.Application method to support the adding of macro items is:

InsertMacroItemInMenu(MacroName as String, MacroItemCaption as String, InMenuTitleCaption as String, [BeforeMenuItemCaption as string]) as long

This function adds the MacroName (which comprises - "<Project>.<Module>.<Subroutine>") using the caption MacroItemCaption to the SA/User menu titled InMenuTitleCaption, before the menu item titled BeforeMenuItemCaption. If no Before item is specified then the item is added to the end of the menu. If the before item is #TOP# then the item inserted at the top of the menu.

The function returns 0 if successful, otherwise none zero.

In other words:

```
Set App = New SA2001.Application

    x = App.InsertMacroItemInMenu("MyProject.MyModule.MySub", "&Test Item",
"&Tools","Ma&cros")
```

This inserts the item 'Test Item' in the Tools menu, before the Macros popup item. When the item is clicked on, the subroutine 'MySub' is executed in the module 'MyModule' in the project 'MyProject'.

For this to succeed the subroutine specified MUST already exist. Also, the captions MUST be specified exactly as in the menu.

For example, Tools is represented as the string &Tools

An ampersand implies an underscore under the following letter. This IS case sensitive.

The subroutine name, should NOT have ()'s following it.

The adding of BMPs to menu items is done by the AssignBMPtoMacroItem method.

AssignBMPtoMacroItem(MacroName as String, BMPFileName as String) as long

This function returns zero if successful, or none zero otherwise.

This should be done prior to adding the macro item to the menu - but need only be done once.

For example:

```
Set App = New SA2001.Application

x = App.AssignBMPtoMacroItem("MyProject.MyModule.MySub", "C:\Piccy.BMP")

x = App.InsertMacroItemInMenu("MyProject.MyModule.MySub", "&Test Item",
"&Tools","Ma&cros")
```

This will add the piccy.bmp to the MySub macro, then add it to the menus as in the previous example. Note that the customize menus (right click on menus) option will now include the Piccy.BMP and the caption '&Test Item', if you wished to put the menu item anywhere else in your menus.

Using the Events

The following SA2001.Application events may assist macro code when deciding when to make menu visibility changes:

- MainMenuUpdate: When the whole menu is affected
- MethodMenuUpdate : When dictionary type menu items are updated
- ToolsMenuUpdate: When tools type menu items are updated
- ReportsMenuUpdate: When report type menu items are updated
- App_ShutDown: When SA shuts down

Events are NO LONGER required to keep user items in the menus. However if you wish to remove items from a menu, the following sections of code provide the outline for such event handling. (Note: The Project is called 'MyProject')

MODULE - AutoExec

```
' Main module which maintains the VBA based event handler
Dim EventHandler As EventCls
' Main Subroutine which starts the event handler
Sub Main()
  Set EventHandler = New EventCls     ' Create an event handler
  Set EventHandler.App = Application  ' Connect the event handler to SA
  ' Perform any additional start up work here
  ' This is a good place to add you BMPs and create your PopUp menus
  InitBMPs
  InitPopUps
End Sub

Sub InitBMPs()
  Dim x As Integer
  x = Application.AssignBMPtoMacroItem("MyProject.MyModule.PRINTDIAGRAMS",
"SAWORD.BMP")
End Sub

Sub InitPopUps()
  Dim x As Integer
  ' NB: Once a popup is created, it remains until it is removed by 'RemovePopupMenu', and
can be added/removed from menus at will.

  ' Create the popup menu with its bitmap
  x = Application.CreatePopUpMenu("Sample Macros","SAWORD.BMP")
  ' Add the item to the popup
  x = Application.InsertMacroItemInMenu("MyProject.MyModule.PRINTDIAGRAMS", "&Print a
Diagram", "Sample Macros")
End Sub
```

CLASS - EventCls

```
Public WithEvents App As Application  ' The application which will raise events

Private Sub Class_Initialize()
  ' No need to do anything
End Sub

Private Sub Class_Terminate()
  ' No need to do anything
End Sub
```

```
Private Sub App_ReportsMenuUpdate()
  ' Reports menu has been redrawn
  Dim x As Long
  x = App.SetSeparatorBefore("&Report Generator...", "&Reports", True)

  ' Insert a menu item in the 'Reports' menu, before the 'Report Generator...' item, called 'Print all Diagrams' which calls PrintDiagrams2
  x = App.InsertMacroItemInMenu("MyProject.MyModule.PRINTDIAGRAMS2", "Print all &Diagrams", "&Reports", "&Report Generator...")

  ' Insert our popup menu 'Sample Macros' in the 'Reports' menu, before the 'Print all Diagrams' item.
  ' NB: You should not keep destroying and creating popups here, unless you have a good reason - do this once at initialization.
  x = App.InsertPopupMenuItemInMenu("Sample Macros", "&Reports", "Print all &Diagrams")
End Sub
```

MODULE - MyModule

```
Public Sub PrintDiagrams()
        ' User Code to run
        MsgBox "Print Diagrams"
End Sub
Public Sub PrintDiagrams2()
        ' User Code to run
        MsgBox "Print Diagrams2"
End Sub
```

How Do I Add a Popup ?

Use the CreatePopUpMenu method with the name you wish to use for your popup. And a BMP if desired. The popup is added to the root collection of popups in SA.

When you leave SA you no longer need your event handler class to remove any added popups using RemovePopUpMenu

How Do I Add Macro Items to a Popup ?

In exactly the same way as you do for adding to SA menus.

How do I Add a Popup to an SA Menu ?

You may add a popup to a Rational System Architect menu in exactly the same way you add a macro item to a menu, except, use InsertPopupMenuItemInMenu and do not specify a macro name.

For example:

InsertPopupMenuItemInMenu(PopUpName as String, InMenuTitleCaption as String, [BeforeMenuItemCaption as string]) as long

How Do I Add a Separator to a Menu Item ?

Use SetSeparatorBefore, provide the menu name and item name, then set True for a separator and false for no separator.

How Do I Remove an Item from a Menu ?

You can remove popups or menu items using the same method: use RemoveItemFromMenu specify the name of the menu and the name of the item.

Note: You can only remove items that you have added. For example, Rational System Architect menu items cannot be removed. The items are not actually removed, they are instead hidden. This allows users to customize the items to be elsewhere in the menu system. Note you can specify “” for the Menu name, as the Menu name is now ignored. The item will be hidden wherever it is in the menu system.

Since all popups and tools which are added to the menu system will remain even after SA has been restarted, it may become necessary to remove a popup menu which is no longer required. The App.RemovePopUpMenu(<PopupName>), method will totally remove the popup from the menu system.

If this method is used in a macro, say at SA shutdown, the popup and all its customization will be removed everytime SA shuts down. This would have the effect, that customization of this popup is not kept between SA sessions.

16

Rational System Architect Relationships

Introduction

This chapter covers all the relationships that exist between objects in the Rational System Architect repository. In the SA Object Browser there exists the enumerated type RELATETYPE. This holds all the possible relationships you can use in your VBA macro.

Each of the diagram, symbol, and definition classes has the method **GetRelatedObjects**. In order to execute this method, the user must specify what relationship exists from those listed in the table below. In addition, the encyclopedia class contains the method, **GetRelationMetric**. The user must specify one of relation types as a required parameter in order to execute the relation metric.

Relation Types

RELATETYPE	Description	Number
RELNULL	Null	0
RELNULL2	Null	1
RELDIAGRAMCON	Diagram Contains Symbols	2
RELCONDIAGRAM	Symbol contained in diagrams	3
RELSHOWTO	Symbol expands to a (child) diagram	4
RELSHOWFROM	A diagram expands from a (parent) symbol	5
RELCONNSTART	A node symbol connects to the start of a line symbol	6
RELSTARTAT	The start of a line connects to a node symbol	7
RELCONNEND	A node symbol connects to the end of a line	8
RELENDAT	The end of a line connects to a node symbol	9
RELFLAGSENDS	A module connects to and sends data via a flag symbol	10
RELFLAGSTR	A flag symbol connects to and receives data from a module	11
RELFLAGRECVS	A module connects to and receives data from a flag symbol	12
RELFLAGEND	A flag symbol connects to and provides data to a module	13
RELDFEELEMENT	An expression uses data (elements or structures)	14

RELATETYPE	Description	Number
RELELEMENTDFE	Data (elements or structures) are used by an expression	15
RELEXPLAINEDBY	A symbol is explained by a comment	16
RELEXPLAINS	A comment explains a symbol	17
RELPARTFULFILLS	A symbol addresses a requirement, test plan, etc.	18
RELPARTFULFILLEDBY	A requirement, test plan, etc. is addressed by a symbol	19
RELCOMPFULFILLS	A symbol is defined by a definition	20
RELDEFINEDBY	A symbol is defined by a definition	20
RELCOMPFULFILLEDBY	A definition defines a symbol	21
RELDEFINES	A definition defines a symbol	21
RELISQUALIFIEDBY	A line or node symbol is "qualified by" a flag symbol	22
RELQUALIFIES	A flag symbol "qualifies" a line symbol (or node symbol)	23
RELISA	A definition "is an instance" of a definition	24
RELINSTBY	A definition is "instantiated by" a definition	25
RELIDENTIFIES	A definition "identifies" another definition	26
RELKEYEDBY	A definition is identified by a definition	27
RELEMBEDS	A node symbol wholly embeds a symbol	28
RELISEMBEDDEDBY	A node symbol is wholly embedded by a node symbol	29
RELISPARENTIN	A definition is a parent of a definition in a parent child relationship	30

RELATETYPE	Description	Number
RELHASPARENTOF	A definition has a parent definition in a parent child relationship	31
RELISCHILDIN	A definition is a child of a definition in a parent child relationship	32
RELHASCHILDOF	A definition has a child definition in a parent child relationship	33
RELCOMPRISES	Definition comprises a definition	34
RELISPARTOF	A definition is part of a definition	35
RELISCATZNOF	Categorization definition categorizes a generic entity definition	36
RELHASCATZN	Generic entity definition has categorization of a categorization definition	37
RELISCATGRYIN	Entity definition is a category in a categorization	38
RELHASCATGRY0F	Categorization has a category of an entity definition	39
RELISCHILDOF	Symbol is the child of another symbol on a hierarchical diagram only	40
RELISPARENTOF	Symbol is the parent of another symbol on a hierarchical diagram only	41
RELISFIRSTCHILDOF	Symbol is the first child of another symbol (e.g., the left-most) on a hierarchical diagram only	42
RELHASFIRSTCHILD	Symbol has another symbol as its first child on a hierarchical diagram only	43
RELISNEXTSIBLING	Symbol is the next sibling of another sibling on a hierarchical diagram only	44

RELATETYPE	Description	Number
RELISPRIORSIBLING	Symbol is the prior sibling of another symbol on a hierarchical diagram only	45
RELISINDEXOF	Data Model - Access Path or Index of Entity or Table	46
RELISINDEXEDBY	A definition is indexed by another definition	47
RELORIGINATESFROM	A definition originated from a definition (e.g. graphic screen diagram)	48
RELISORIGINOF	A definition is origin of a definition	49
RELISBASEDON	A definition is based on a definition (usually a data element)	50
RELISBASISFOR	A definition is basis for derived definition	51
RELISLINKEDTO	A symbol is linked to another symbol	52
RELISLINKEDWITH	A symbol is linked with another symbol	53
RELUSER AND RELUSERCOMPLEMENT	User and user complement	54 thru 83
RELPOPKIN AND RELPOPKINCOMPLEMENT	IBM and IBM complement	84 thru 111
RELREPRESENTS	Explorer diagram symbol represents an object	112
RELISPRESENTEDBY	Object is represented by an Explorer diagram symbol	113
RELPOPKIN AND RELPOPKINCOMPLEMENT	IBM and IBM complement	114 thru 125
RELUSER AND RELUSERCOMPLEMENT	User and user complement	126 thru 135
RELLINKS TO	Object is linked to object in DOORS	136

RELATETYPE	Description	Number
RELISLINKEDFROM	Object is linked from object in DOORS	137
RELISTOBESENTTO	Object is to be sent to DOORS module	138
RELISTORECEIVE	DOORS module is to receive object	139
RELHASBEENSENTTO	Object has been sent to DOORS module	140
RELHASRECEIVED	DOORS module has received object	141

17

Rational System Architect Field Types

Introduction

This chapter covers all the field types available to the user that are listed in the Rational System Architect Object Browser under the enumerated type FLDTYPE. These constants determine the format in which data is returned. In most cases Rational System Architect has set the Field Type internally. This enumerated type is used as a required parameter for the GetRelationMetric method of the Encyclopedia class. It is also an optional parameter in the GetMetric methods of the Diagram, Symbol, and Definition classes.

The table below provides a complete listing of all the Field Types and their descriptions.

Field Types

FLDTYPE	Number	Description
FLDTYPAUTO	65	This is the default field selected by Rational System Architect internally.
FLDTYPCHARACTER	67	Holds a string up to 256-characters
FLDTYPDATE	68	This is a date field (e.g. MM/DD/YYYY)
FLDTYPLOGICAL	76	Boolean field
FLDTYPMEMO	77	Memo fields store large blocks of text up to 4,095 characters.
FLDTYPNUMERIC	78	Specifies that the field must contain values that are numbers.
FLDTYPTIME	84	This is a time field (e.g. hours:minutes:seconds)

18

Rational System Architect Errors

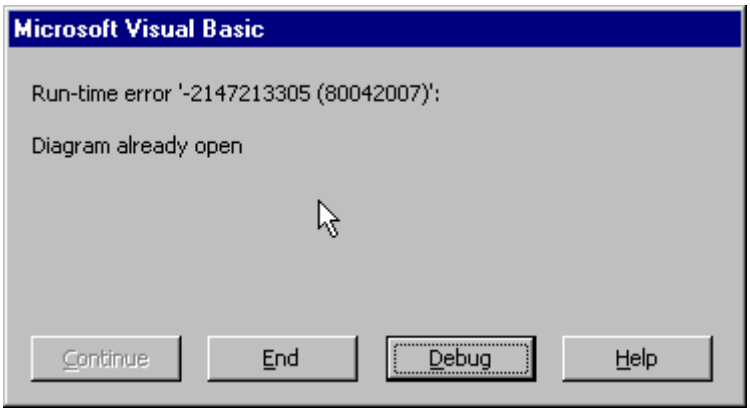
Introduction

This chapter focuses on the errors generated by Rational System Architect that can be trapped during code execution. It does not look at errors within the written code (an example of which is the Visual Basic error 91 “Object Variable or With block variable not set” where an instance of the variable has not been created).

Handling Errors

The properties and methods in the Rational System Architect Object Model will execute their code in a consistent way but unfortunately the base data may not always be as expected and errors may occur during the natural operation of a VBA macro. A number of issues may arise during normal operation and these need to be trapped and handled otherwise the macro will terminate abruptly without finishing the code execution.

When an error occurs during code execution the following dialog is shown. The method **Diagram.Show** has been executed to open the current diagram graphic. However, the diagram has already been opened at some other point in the code and so an error occurs.



The user has the option of clicking on End to end the program at the current point or Debug which opens the VBA editor interface and highlights the position at which the error occurred in the code. The user may get further explanation by clicking the Help button.

Rational System Architect Errors

Rational System Architect has a number of errors that can be identified. VBA can follow these errors when found in a macro and pass a message to the VBA error handler, which then presents the error to the user. The different errors catch different problems. When a method is executed, it returns a value to VBA indicating the status of the call; if an exception occurs, then VBA reports the error. The enumerated type SA2001Errors contains a complete list of all Rational System Architect Errors.

Error	Number	Description
SAERR_BADDDID	8195	
SAERR_DIAGRAMNOTOPEN	8198	Diagram is not open. Either the diagram was never open or the **.Hide** method was previously invoked
SAERR_DIAGRAMNOTSAME	8201	
SAERR_DIAGRAMOPEN	8199	Diagram is already open. Either the diagram has been previously opened or the **.Show** method was previously invoked.
SAERR_ENUMVARIANTERROR	8193	
SAERR_INVALIDCLASS	8202	
SAERR_INVALIDOBJECT	8194	
SAERR_INVALIDPROPERTY	8204	
SAERR_INVALIDTYPE	8207	
SAERR_NOTIMPLEMENTED	8196	
SAERR_OBJECTDOESNOTEXIST	8197	Either the object was never instantiated or was previously deleted.
SAERR_OBJECTISLOCKED	8205	Either the OpenLock method was previously invoked or the object was previously checked out or frozen by another user.

Error	Number	Description
SAERR_OBJECTNOTFOUND	1025	
SAERR_OPENEDASREADONLY	8206	The object has been opened as Read-only.
SAERR_REQUIREDPROPERTYABSENT	8192	The referenced property is either blank or does not exist.
SAERR_SA20001_IMF_ERROR	4096	An SAIMF error has occurred.
SAERR_SANOTRUNNING	1024	Rational System Architect has either never been started up or was previously shutdown.
SAERR_SYMBOLHASNODIAGRAM	8200	The symbol is not referenced by any diagram.
SAERR_TOOMANYOBJECTS	8203	

19

IBM support

Introduction

There are a number of self-help information resources and tools to help you troubleshoot problems. If there is a problem with your product, you can:

Refer to the release information for your product for known issues, workarounds, and troubleshooting information.

Check if a download or fix is available to resolve your problem.

Search the available knowledge bases to see if the resolution to your problem is already documented.

If you still need help, contact IBM® Software Support and report your problem.

Topics in this chapter	Page

Contacting IBM Rational Software Support

If you cannot resolve a problem with the self-help resources, contact IBM® Rational® Software Support.

Note: If you are a heritage Telelogic customer, you can find a single reference site for all support resources at http://www.ibm.com/software/rational/support/telelogic/

Prerequisites

To submit a problem to IBM Rational Software Support, you must have an active Passport Advantage® software maintenance agreement. Passport Advantage is the IBM comprehensive software licensing and software maintenance (product upgrades and technical support) offering. You can enroll online in Passport Advantage at http://www.ibm.com/software/lotus/passportadvantage/howtoenroll.html

- To learn more about Passport Advantage, visit the Passport Advantage FAQs at http://www.ibm.com/software/lotus/passportadvantage/brochures_faqs_quickguides.html.
- For further assistance, contact your IBM representative.

To submit a problem online (from the IBM Web site) to IBM Rational Software Support:

- Register as a user on the IBM Rational Software Support Web site. For details about registering, go to http://www.ibm.com/software/support/.
- Be listed as an authorized caller in the service request tool.

Other information

For Rational software product news, events, and other information, visit the IBM Rational Software Web site: http://www.ibm.com/software/rational/.

Submitting problems

To submit a problem to IBM Rational Software Support:

1. Determine the business impact of the problem. When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem.

To determine the severity level, use the following table.

Severity	Description
1	The problem has a critical business impact: you are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
2	The problem has a significant business impact: the program is usable, but it is severely limited.
3	The problem has some business impact: the program is usable, but less significant features (not critical to operations) are unavailable.
4	The problem has minimal business impact: the problem causes little impact on operations or a reasonable circumvention to the problem was implemented.

2. Describe the problem and gather background information. When you describe the problem to IBM, be as specific as possible. Include all relevant background information so that IBM Rational Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?

- To determine the exact product name and version, use the option applicable to you:
    - Start the IBM Installation Manager and click **File** > **View Installed Packages**. Expand a package group and select a package to see the package name and version number.
    - Start your product, and click **Help** > **About** to see the offering name and version number.

- What is your operating system and version number (including any service packs or patches)?

- Do you have logs, traces, and messages that are related to the problem symptoms?

- Can you recreate the problem? If so, what steps do you perform to recreate the problem?

- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, or other system components?

3. Are you currently using a workaround for the problem? If so, be prepared to describe the workaround when you report the problem.

4. Submit the problem to IBM Rational Software Support in one of the following ways:

    - Online: Go to the IBM Rational Software Support Web site at https://www.ibm.com/software/rational/support/. In the Rational support task navigator, click **Open Service Request**. Select the electronic problem

reporting tool, and open a Problem Management Record (PMR) to describe the problem.

- For more information about opening a service request, go to http://www.ibm.com/software/support/help.html.

- You can also open an online service request by using the IBM Support Assistant. For more information, go to http://www.ibm.com/software/support/isa/faq.html.
- By phone: For the phone number to call in your country or region, visit the IBM directory of worldwide contacts at http://www.ibm.com/planetwide/ and click the name of your country or geographic region.

- Through your IBM Representative: If you cannot access IBM Rational Software Support online or by phone, contact your IBM Representative. If necessary, your IBM Representative can open a service request for you. For complete contact information for each country, visit http://www.ibm.com/planetwide/.

20

Appendix:

Introduction

This chapter contains information about the legal uses and trademarks of IBM® Rational® System Architect®.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send written license inquiries to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send written inquiries to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or

implied warranties in certain transactions. Therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Intellectual Property Dept. for Rational Software
IBM Corporation
1 Rogers Street
Cambridge, MA 02142
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been

estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Copyright license

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2000 2009.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.html

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product or service names mentioned may be trademarks or service marks of others.