Using the Rose Extensibility Interface

Rational Rose®

VERSION: 2001A.04.00

PART NUMBER: 800-024466-000

support@rational.com
http://www.rational.com

Rational®
the e-development company™

COPYRIGHT NOTICE

U.S. GOVERNMENT RIGHTS NOTICE

TRADEMARK NOTICE

Printed in the U.S.A.

Contents

Preface xv
Audience xv
Other Resources xv
Related Documentation xvi
File Names xvi
Contacting Rational Technical Publications xvi
Contacting Rational Technical Support xvii
1 Basic Extensibility Concepts 1
Contents 1
Rational Rose Extensibility 1
The REI Model and Rose Extensibility 1
Rose Scripting 3
Rose Automation 4
Rose Add-In Manager 4
Default Properties and Property Sets 5
Rose Extensibility Type Libraries 5
2 Customizing Rational Rose Menus 7
Contents 7
Extending Rational Rose Menus 7
Customizing Rose Main Menus 7
Procedure 8
Adding Entries to a Rose Menu File 9
Menu File Keywords 9
Menu Actions 10
Menu File Variables and Modifiers 11
Syntax Rules for Rose Menu File Entries 14
Adding Scripts to a Rose Menu 15
Adding or Editing the Virtual Path for Scripts 16
Sample Rose Menu File 17
Customizing Rose Shortcut Menus 19
Benefits 19
Limitations 20
Key Terms and Concepts 20
Language-Dependent 20
Language-Neutral 21

Language Add-In . . . 21
Non-Language Add-In . . . 22
Behind the Scenes of Shortcut Menus . . . 22
How Rose Formats and Displays Shortcut Menu Items . . . 22
Shortcut Menu Scenarios . . . 23
Shortcut Menu Design Considerations . . . 26
Procedure . . . 26
Adding Menu Items to the Shortcut Menu . . . 27
Working with Shortcut Menu Items . . . 27
Working with the Shortcut Menu Item Collection . . . 27
Editing Shortcut Menu Items . . . 28
Changing the State of a Shortcut Menu Item . . . 28
Sample Shortcut Menu Implementation Code . . . 28
Sample Rose Script Shortcut Menu Code . . . 30
3 Using the REI to Work with Rational Rose . . . 35
Contents . . . 35
Introduction . . . 35
Getting the Rose Application Object . . . 36
Using Rose Script . . . 36
Using Rose Automation . . . 36
Associating Files and URLs with Classes . . . 37
Managing Default Properties . . . 37
Adding a Property to a Set . . . 39
How To . . . 39
Example . . . 39
Notes on the Example . . . 40
Creating a New Property . . . 40
How To . . . 40
Example . . . 40
Notes on the Example . . . 41
Deleting Model Properties . . . 41
Getting Model Properties . . . 41
Setting Model Properties . . . 41
Setting Model Properties Using OverrideProperty . . . 42
How To . . . 42
Example . . . 42
Notes on the Example . . . 42
Setting Model Properties Using InheritProperty . . . 43
How To . . . 43

Example 43
Notes on the Example 43
Creating a New Property Set 43
Cloning a Property Set 44
How To 44
Example 44
Notes on the Example 45
Deleting a Property Set 45
How To 45
Example 45
Notes on the Example 46
Getting and Setting the Current Property Set 46
How To 46
Example 46
Notes on the Example 47
Creating a User-Defined Property Type 47
How To 47
Example 47
Notes on the Example 48
Creating a New Tool 48
Placing Classes in Categories 48
Using Type Libraries for Rose Automation 49
How To 49
Example 49
Working with Controllable Units 49
Working with Rose Diagrams 50
Getting an Element from a Collection 51
Accessing Collection Elements by Count 51
How To 51
Example 51
Accessing Collection Elements by Unique ID 51
How To 51
Example 52
Accessing Collection Elements by Name 52
How To 52
Example 52
4 Using the Rational Rose Script Editor 53
Contents 53
The Rose Script Editor 53

The Script Editor Window 54
Opening a Script 54
Creating New Rose Scripts 55
Creating a New Script from Scratch 55
Creating a New Script from an Existing Script 55
Selecting a Font for the Script Editor 55
Moving the Insertion Point in a Script 56
Moving the Insertion Point with the Mouse 56
Moving the Insertion Point to a Specified Line in Your Script 56
Selecting Text 57
Selecting Text with the Mouse 57
Selecting Text with the Keyboard 58
Selecting an Entire Line 58
Deleting, Cutting, Copying, and Pasting Text 59
Deleting Text 59
Cutting a Selection 59
Copying a Selection 59
Pasting the Contents of the Clipboard into Your Script 59
Adding Comments to a Script 59
Adding a Full-Line Comment 59
Adding a Comment at the End of a Line of Code 60
Finding and Replacing Text 60
Finding Specified Text 60
Replacing Specified Text 61
Running, Pausing, and Stopping Your Script 62
Running Your Script 62
Pausing an Executing Script 62
Stopping an Executing Script 62
Tracing Script Execution 63
Stepping Through Your Script 63
Displaying the Calls Dialog Box 63
Setting and Removing Breakpoints 64
Starting Debugging Partway Through a Script 64
Continuing Debugging at a Line Outside the Current Subroutine 65
Debugging Selected Portions of Your Script 65
Removing a Single Breakpoint Manually 66
Removing All Breakpoints Manually 66
Working with Watch Variables 66

Adding Watch Variables 66
Selecting Variables on the Watch List 68
Deleting Watch Variables 68
Modifying the Value of Variables on the Watch Variable List 68
Compiling Your Script 69
Using Interscript Calls 70
Guidelines for Using a Script to Call Another Script 70
Debugging Interscript Calls 70
Working with the Dialog Editor 70
Inserting a Dialog Box into Your Script 70
Editing an Existing Dialog Box 71
Displaying and Adjusting the Grid 71
Changing Titles and Labels 73
Assigning Accelerator Keys 73
Capturing Standard Windows Dialog Boxes 73
Testing Your Dialog Boxes 74
Incorporating Dialog Boxes or Controls into Your Script 75
Selecting Controls 76
Selecting Dialog Boxes 76
Repositioning Items 77
Repositioning Items with the Mouse 77
Repositioning Items with the Arrow Keys 77
Repositioning Dialog Boxes with the Dialog Information Dialog Box 77
Repositioning Controls with the Dialog Information Dialog Box 78
Resizing Items 78
Resizing Items with the Mouse 78
Resizing Items with the Information Dialog Box 78
Resizing Selected Items Automatically 79
Adding Controls 79
Duplicating Controls 80
Adding Pictures to a Dialog Box 80
Adding Pictures from Files 80
Adding Pictures from Picture Libraries 81
Pasting Items into Dialog Editor 81
Pasting Existing Dialog Boxes into the Dialog Editor 81
Pasting Controls from Existing Dialog Boxes into the Dialog Editor 82
Displaying the Information Dialog Boxes 82
Displaying the Information Dialog Boxes for Dialog Boxes 82
Attributes You Can Adjust with the Dialog Box Information Dialog Box 83

Displaying the Information Dialog Boxes for Controls 83
Attributes You Can Adjust with the Information Dialog Boxes for Controls. . 84

A Rational Rose Script Editor Shortcuts 89
Contents 89
General Shortcuts 89
Navigating Shortcuts 90
Editing Shortcuts 90
Debugging Shortcuts 91
File Menu Shortcuts 92
Edit Menu Shortcuts 92
Debugger Menu Shortcuts 93

B Developing Add-Ins for Rational Rose 95
Contents 95
Introduction 95
Why Create Add-Ins? 96
Types of Add-Ins 97
What Is in an Add-In? 97
Main Menus 97
Shortcut Menu 98
Custom Specifications 98
Properties 98
Data Types 98
Stereotypes 98
Online Help 98
Context-sensitive Help 99
Registering for Events 99
Functionality 99
UNIX vs. Windows 99
Creating Portable Add-Ins 100
How to Develop Add-Ins 101
Customizing Main Menus 102
Customizing the Shortcut Menu 103
Creating Custom Specifications 103
Customizing Properties 103
Design Considerations 103
Information in Property Files 104
Format for Property Files 105

Sample Property File 109
Creating Property Files 110
Testing Property Files 110
Customizing Data Types 111
Customizing Stereotypes 111
Steps for Creating Add-In Stereotypes 112
Additional Online Help 116
Adding Online Help for Your Add-In 116
Additional Context-sensitive Help 117
Main Menu Items 117
Model Properties 118
User Manuals 118
Registering for Events 118
Interface vs. Script Events 119
What Events Are Available? 119
How to Add Events to Your Add-In 120
Updating the Registry 123
Registry Entries 123
Registering Custom Stereotypes 125
Updating the Registry During Installation 125
Registry File Anatomy 126
Installing, Setting Up, and Uninstalling Your Add-In 126
Installation Reminders 127
Installing Add-Ins 128
Uninstalling Add-Ins 128
Activating and Deactivating Add-Ins 128

Index 129

Figures

Figure 1 Rose Extensibility Model — Logical View 2
Figure 2 Rose Application and Extensibility Components 3
Figure 3 Adding Virtual Path for Scripts 16
Figure 4 Sample Rose Menu File 17
Figure 5 Sample Code for Shortcut Menus 30
Figure 6 Property Specification Editor 38
Figure 7 Rose Script Editor 54
Figure 8 Goto Line Dialog Box 57
Figure 9 Selected Script Text 58
Figure 10 Find Dialog Box 60
Figure 11 Replace Dialog Box 61
Figure 12 Script Calls Dialog Box 64
Figure 13 Add Watch Dialog Box 66
Figure 14 Modify Variable Dialog Box 69
Figure 15 Grid Dialog Box 72
Figure 16 Dialog Editor with Grid Displayed 72
Figure 17 Capturing a Dialog Box 74
Figure 18 Sample Dialog Box in Basic Script 76
Figure 19 Dialog Box Information Dialog Box 83
Figure 20 Control Information Dialog Box 84
Figure 21 Rose Add-Ins Architecture 96
Figure 22 Sample Custom Properties 109
Figure 23 OLEServer Windows Registry Entry 120
Figure 24 Windows Registry Entries for Rose Events 121
Figure 25 Sample Add-In Event Handler 122
Figure 26 Windows Registry Entries for an Add-In 123
Figure 27 Registry Entry for a Custom Stereotype Configuration File 125

Tables

Table 1	Menu File Keywords	9
Table 2	Menu Actions	10
Table 3	Menu File Variables	12
Table 4	Menu File Modifiers	12
Table 5	Displaying Shortcut Menu Items	23
Table 6	Sample Watch Expressions	67
Table 7	General Shortcuts	89
Table 8	Navigating Shortcuts	90
Table 9	Editing Shortcuts	90
Table 10	Debugging Shortcuts	91
Table 11	File Menu Shortcuts	92
Table 12	Edit Menu Shortcuts	92
Table 13	Debugger Menu Shortcuts	93
Table 14	UNIX vs. Windows	99
Table 15	Property File Data Types	107

Preface

Using the Rose Extensibility Interface describes the Rational Rose Extensibility Interface (REI) and provides procedures for:

- Customizing and extending Rose menus
- Customizing and extending Rose using the REI
- Working with the Rose Script Editor, which is the scripting environment for working with the REI

Audience

This manual is intended for scripters and add-in developers who want to customize and extend Rose. It assumes that you are familiar with the Windows 95, Windows 98, Windows NT 4.0, or Windows 2000 operating environment; object oriented design concepts; and how to use Rose.

Other Resources

- Online Help is available for the Rose Extensibility Interface.

  From Rose, click **Help > Contents and Index > Contents > Rational Rose Extensibility Interface**.
- All manuals are available online, either in HTML or PDF format. The online manuals are on the *Rational Solutions for Windows Online Documentation* CD.
- To purchase additional printed documentation for Rational products, see http://www.rational.com/documentation.
- For more information on training opportunities, see the Rational University Web site: http://www.rational.com/university.

Related Documentation

For additional resources, refer to the Using Rose guide and online Help. If you are new to Rose, visual modeling, or the Unified Modeling Language (UML), you may also want to read the book, *Visual Modeling with Rational Rose and UML*.

File Names

Where file names appear in examples, Windows syntax is depicted. To obtain a legal UNIX file name, eliminate any drive prefix and change the backslashes to slashes:

c:\project\username

becomes

/project/username

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our Technical Documentation Department at techpubs@rational.com.

Contacting Rational Technical Support

If you have questions about installing, using, or maintaining this product, contact Rational Technical Support.

Your Location	Telephone	Fax	E-mail
North America	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4545-201 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

Note: When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously-reported problem)

Basic Extensibility Concepts 1

Contents

This chapter is organized as follows:

- *Rational Rose Extensibility* on page 1
- *The REI Model and Rose Extensibility* on page 1
- *Rose Scripting* on page 3
- *Rose Automation* on page 4
- *Rose Add-In Manager* on page 4
- *Default Properties and Property Sets* on page 5
- *Rose Extensibility Type Libraries* on page 5

Rational Rose Extensibility

Rational Rose provides several ways for you to extend and customize its capabilities to meet your specific software development needs. You can:

- Customize Rose menus.
- Automate manual Rose functions with Rose Scripts (for example, diagram and class creation, model updates, and document generation).
- Execute Rose functions from within another application by using the Rose Automation object (RoseApp).
- Access Rose classes, properties, and methods right within your software development environment by including the Rose Extensibility Type Library in your environment.
- Activate Rose add-ins using the Add-In Manager.

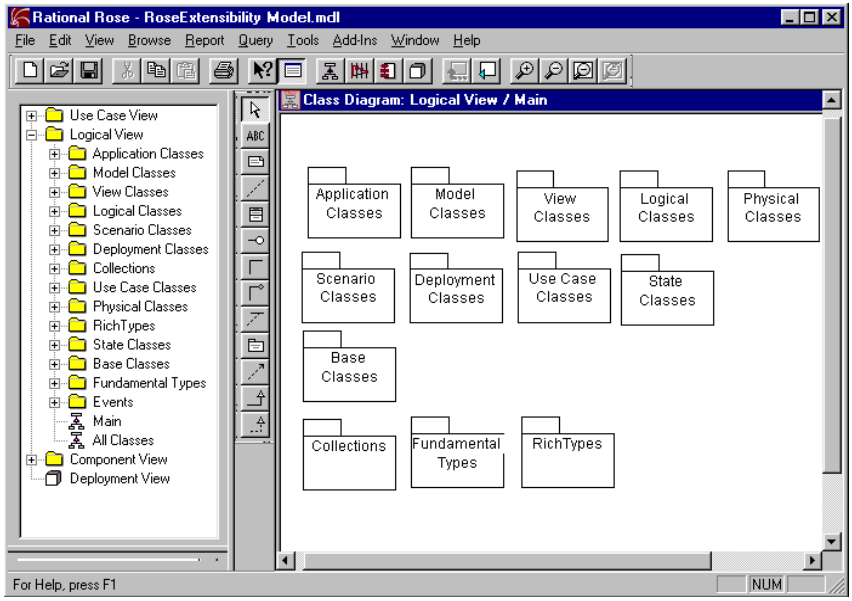
The REI Model and Rose Extensibility

The purpose of Rose is to enable component-based software development. As you would expect, the Rose application is itself component based, and is defined in the Rose Extensibility Interface (REI) Model.

The REI Model is essentially a metamodel of a Rose model, exposing the packages, classes, properties, and methods that define and control the Rose application and all of its functions.

Figure 1 on page 2 shows the logical packages that comprise the Rose Extensibility Interface Model. Refer to the online Help for details on the classes contained in each package, and the properties and methods defined for each class.

Figure 1 Rose Extensibility Model — Logical View



You communicate with the Rose Extensibility Interface through Rose Scripts or through Rose Automation. In either case, you will use the REI calls defined in the Rose Extensibility Model and described in the online Help.

Figure 2 on page 3 shows the components of Rose and the Rose Extensibility Interface, and illustrates the relationships between them. These components are:

- Rose Application

  The Rose Extensibility objects that interface to Rose's application functionality.

- Rose Extensibility Interface

  This is the common set of interfaces used by Rose Script and Rose Automation to access Rose.

- Rose Script

  The set of Rose Script objects that allow Rose Scripts to automate Rose functionality.

- Rose Automation

  The set of Rose Automation objects that allow Rose to function as an OLE automation controller or server.

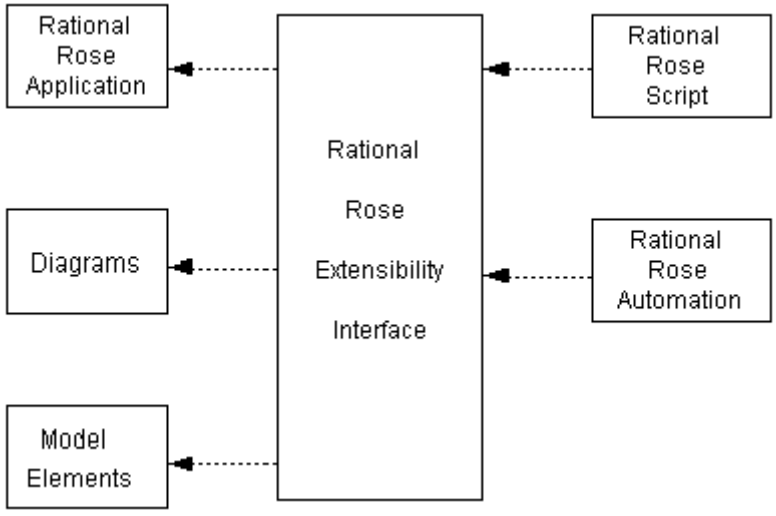
- Diagrams

  The Rose Extensibility objects that interface to Rose's diagrams and views.

- Model Elements

  The Rose Extensibility objects that interface to Rose's model elements.

Figure 2 Rose Application and Extensibility Components



Rose Scripting

The Rose Scripting language is an extended version of the Summit BasicScript language. The Rose extensions allow you to automate Rose-specific functions and, in some cases, perform functions that are not available through the Rose user interface.

The Rose Script Editor runs in the Rose environment and provides access to the scripting environment. Start the script editor by clicking either **Tools > New Script** or **Tools > Open Script**.

Rose provides a set of sample scripts that you can use as a base from which to create your own scripts.

- Check the **Scripts** folder in your Rose installation directory for the complete list of available scripts.
- Use the Rose Script Editor (click **Tools > Open Script**) to view a sample script. If you want to edit the script, click **File > Save Script As** to create a copy for your own use, leaving the sample intact.

Use the online BasicScript and Rose Script Language References for complete script language information.

Rose Automation

Rose Automation allows you to integrate other applications with Rose in two ways:

- Using Rose as an automation controller, you can call an OLE automation object from within a Rose script. For example, a Rose script can use OLE automation to execute functions in applications such as Word and Visual Basic.
- Using Rose as an automation server, you can call its OLE automation object from within other OLE-compliant applications.

Rose Automation is accessible to automation controller environments, such as Visual Basic, EXCEL, Summit BasicScript, Softbridge Basic Language, C, C++, and others.

Note: You may need to adapt the syntax listed for each REI property and method to your particular programming language. If the listed syntax does not meet your needs, consult your programming environment's Help, programming language books, and outside documentation on the subject.

Rose Add-In Manager

The Rose Add-In Manager provides you with the facilities required to install extensions you create as add-in components in the Rose environment.

In the extensibility environment, you can manipulate add-ins using calls to the RoseAddInManager object.

Default Properties and Property Sets

Each Rose model has its own default properties. These default properties are defined in a property file and are grouped into sets based on:

- Type of model element

  Class, component, relation, attributes, operations, and so on; the objects that make up the model.

- Tool

  Corresponds to a tab in the property specification. A tool can be a programming language tool, such as Java or C++; a database tool, such as Oracle8; a user-defined add-in to Rose; or some other tool.

- Properties

  The actual properties and property values defined in the set; these must be appropriate to the model element and tool for which they are being defined.

Note: You can define multiple sets of default properties for the same tool and model element. For example, you might want one set of properties for a class with a stereotype of Actor and a different set of properties for a class with a stereotype of Interface. Both of these sets are still considered default properties in that they are predefined for the model. Defining multiple sets saves you work by minimizing the need to override properties later.

Rose Extensibility Type Libraries

Loading a type library for Rose automation allows you to use Rose class names to access the Rose Extensibility Interface from your programming environment.

For example, if you are working in Visual Basic, instead of using the Basic object type Object, you can use the name of the actual Rose class. You can also check the syntax of the properties and methods at compile time (early binding) instead of when the code is executed (late binding).

If you are working in Visual C, you can import RationalRose.tlb into an MFC project. This generates ColeDispatchDriver subclasses for each REI class, and methods allowing access to REI properties and methods.

Important: When you specify a Rose class name in an automation environment, you must add the prefix **Rose** to the class name, unless the class name itself contains the word Rose already. (For example, the Rose class, RoseItem, does not require a prefix.) This prefix prevents class name conflicts across applications.

For example, in Rose Script, the syntax for declaring a Category variable is:

```
Dim theCategory As Category
```

In Rose Automation, the syntax for declaring a Category variable is:

```
Dim theCategory As RoseCategory
```

For details on using type libraries in any automation environment, refer to the documentation for your particular programming environment.

Customizing Rational Rose Menus 2

Contents

This chapter is organized as follows:

- *Extending Rational Rose Menus* on page 7
- *Customizing Rose Main Menus* on page 7
- *Customizing Rose Shortcut Menus* on page 19

Extending Rational Rose Menus

Using the Rational Rose Extensibility Interface, you may add your own menu options to one of Rose's menus (for example, **File** and **Edit**). You can also add your own menu options to the Rose shortcut menu (displayed when you right-click).

This chapter explains how to customize the:

- Rose main menus.
- Rose shortcut menu.

Customizing Rose Main Menus

You can extend or customize Rose menus by updating the Rose menu file, which Rose reads during startup.

You can extend Rose menus by adding:

- Submenus.
- Menu items that execute any of the following:
  - Rose primitives
  - Rose scripts
  - System commands
  - External programs

- Menu separators (lines between menu items, used to group similar menu items).

Note: You can add information to existing menus (for example, **File** and **Edit**); however, you cannot add new menus to the Rose menu bar.

The content of Rose menus is defined in the Rose.mnu file. If you want to customize Rose menus, you must edit this file.

While you cannot add new menus to the Rose menu bar, you can add commands to the existing Rose menus.

Use the procedures, commands, and syntax described in this chapter to add Rose menu commands that:

- Execute a program or shell script.
- Execute a Rose script.
- Load or save controllable units.
- Display a dialog box for user input.
- Change write protection for a controllable unit.
- Execute an interface in a COM server (for example, from your add-in).

Procedure

The following procedure outlines the general steps for customizing Rose menus.

The subsections following the procedure provide information on command syntax, variables, and modifiers to use as you complete the procedure.

Check the sample menu file at the end of this chapter for a complete example that illustrates how to put the various menu elements together into a working menu file.

1. Using any text editor, open the Rose.mnu file. (The file resides in the directory where Rose is installed.)
2. Add entries to Rose.mnu for any or all of the following:
   - Submenus
   - Menu options
   - Menu separators

   Note: Pay close attention to the syntax rules that apply to your entries to the Rose menu file. For example, the syntax of the menu specifications includes opening and closing braces. You must include these braces in your specifications or they will not work properly. For complete details, see *Syntax Rules for Rose Menu File Entries* on page 14.
3. If the menu item executes a script, add or edit Rose's virtual path for scripts, unless one is already defined.

4 Save the file.

- To create another menu file while leaving the Rose.mnu file intact, save the file under a different name. (Recommended)
- To overwrite the file, save it as Rose.mnu.

Adding Entries to a Rose Menu File

Using any text editor and the following information, you can add menu entries to the Rose menu file. The entries will appear on the Rose menu in the order in which you specify them.

As you add menu entries, you will specify:

- Keywords that determine what to add to the menu (a submenu, a menu option, a separator).
- Menu actions that specify what action to take when the menu item is selected.
- Arguments that further define a menu action, or that determine the conditions under which a menu command is enabled or disabled in Rose.

Remember to follow all of the syntax rules as described in *Syntax Rules for Rose Menu File Entries* on page 14. For example, the syntax of the menu specifications includes opening and closing braces. You must include these braces in your specifications or they will not work properly. Remember that each opening brace ({) requires a corresponding closing brace (}).

Menu File Keywords

Table 1 on page 9 describes the valid keywords for your entries in the Rose menu file.

Table 1 Menu File Keywords

Keyword	Description
Menu *RoseMenu*	Enter the **Menu** keyword, followed by the Rose menu name to indicate the name of the menu being extended. For example, enter **Menu Tools** as the first line of an entry that extends the **Tools** menu.
Menu "*Menu Text*"	Enter the **Menu** keyword, followed by a text string to indicate the name of a submenu being added to the menu. Note that quotation marks are required if the text string contains spaces. For example, enter **Menu "My Scripts"** to add a submenu called **My Scripts**.

Table 1 Menu File Keywords (continued)

Keyword	Description
Separator	Enter the **Separator** keyword to add a separator to a list of menu options. Remember the placement of the Separator keyword controls the placement of the separator line on the menu.
Option "*Command text*"	Enter the **Option** keyword, followed by a text string to indicate the name of the menu command being added to the menu. Note that quotation marks are required if the text string contains spaces. For example, enter **Option "Run My Script"** to add a menu command called **Run My Script**.

Menu Actions

An action defines the result of activating a menu entry. The required arguments can be supplied as keywords, constants, variables, or variables with modifiers. Table 2 on page 10 describes the valid menu actions for your entries in the Rose menu file.

Table 2 Menu Actions

Action	Result
Block	Displays a modal dialog box with 'arg' as its prompt. Used following 'exec' and an action such as the **Roseload** command to suspend the following action until the user chooses to continue.
RoseScript *Script-Path-and-Name*	Executes a source or compiled image of a script. You can specify the script name without its extension. The **Rosescript** command will search for the source script first and execute it if found. If not found, it will search for and execute the compiled script.
Exec *program-name* [*arg2* [*arg3* ...[*arg10*]]]	Executes the program or shell script contained in the file designated by *program-name*. (If the program is not located in the current directory, it must be in a directory in the execute path.) If the final argument is of the form '-F<filename>' then a file named <filename> is created (if it does not already exist). All arguments, except the last one are written to the file, and <filename> is passed as the sole argument to 'program'. **Note:** ▪ F must be uppercase. ▪ It is up to 'program' to delete the file. ▪ To pass a string beginning with '-F' as the final parameter of an exec action, use '--F' instead. (The character '^' does NOT work in this case.)

Table 2 Menu Actions (continued)

Action	Result
Roseload *ControlledUnit*	Loads the designated controlled unit(s) from the associated file.
Rosesave *ControlledUnit*	Saves the designated controlled unit(s) to the associated file.
Updateaccess *ControlledUnit*	Sets the write protection for the controlled unit(s) to that of their corresponding files.
InterfaceEvent *ToolDisplayName interface*	Executes the specified interface in the specified add-in's registered COM object. You are not limited to Rose events. You may specify custom interfaces from your OLE server. Note that quotation marks are required if the *ToolDisplayName* contains spaces. Rose looks for the add-in whose registry key, ToolDisplayName, matches the *ToolDisplayName* argument. If the ToolDisplayName registry key is blank, Rose then looks for a match to the ToolName registry key. For more information on registry settings for Rose add-ins, see *Updating the Registry* on page 123. Examples: ▪ InterfaceEvent C++ OnBrowseHeader When the user selects the menu option corresponding to this menu action, Rose executes the OnBrowseHeader method in the add-in's OLE server whose ToolDisplayName (or, if ToolDisplayName is blank, ToolName) registry key is “C++”. ▪ InterfaceEvent All OnBrowseHeader When the user selects the menu option corresponding to this menu action, Rose executes the OnBrowseHeader method in all active add-ins' OLE servers. ▪ InterfaceEvent “My AddIn” CheckFormat When the user selects the menu option corresponding to this menu action, Rose executes the add-in's custom CheckFormat method in the add-in's OLE server whose ToolDisplayName (or, if ToolDisplayName is blank, ToolName) registry key is “My AddIn”.

Menu File Variables and Modifiers

Rose provides a set of variables that correspond to various Rose model items. You can use these variables in conjunction with a set of modifiers to determine the conditions under which menu items are enabled or disabled, as well as to specify specific menu actions.

The format for specifying variables with modifiers is:

`variable``[:`*`mod1`*`[:`*`mod2`*`[...[:`*`mod10`*`]]]`

Variables

Table 3 on page 12 lists the set of variables that are valid for extending Rose menus.

Table 3 Menu File Variables

Variable	Description
%all_units	List of controlled units in all models.
%current_diagram	Name of the current diagram.
%true	Boolean value `TRUE`.
%false	Boolean value `FALSE`.
%model	Name of the current model.
%selected_items	List of model elements selected in the current diagram.
%selected_units	List of controlled units selected in the current diagram.
%uname	Use in place of %selected_units:first:elide. See *Modifiers* for information on first and elide.
%ufile	Use in place of %selected_units:first:file. See *Modifiers* for information on first and file.

Modifiers

Table 4 on page 12 lists the set of modifiers that are valid for use with variables to extend Rose menus.

Table 4 Menu File Modifiers

Modifier	Description
allfiles	Applied to a unit or item name or a list of unit or item names, evaluates to a string that contains the list of the corresponding header and source file names.
basename	Applied to a path, evaluates to a string that contains the file name portion of the path. Applied to a list of paths, evaluates to a string that contains a list of file names. Each file name is extracted from its corresponding path.

Table 4 Menu File Modifiers (continued)

Modifier	Description
codefile	Applied to a unit or item name or a list of unit or item names, does one of the following: ▪ Evaluates to a string that contains the complete path of the codefile attribute associated with the unit. ▪ Evaluates to a string that contains the name of the controlled unit in which the item is located. Applied to a list, evaluates to a string that contains the list of corresponding file names.
directory	Applied to a path which resolves to a file, evaluates to a string that contains the directory portion of the path. Applied to a path which resolves to a directory, evaluates to a string that contains that path—no modification is performed. Applied to a list of paths, evaluates to a string that contains a list of directories. Each directory is extracted from its corresponding path using the preceding rules.
elide	Applied to a unit or item name, evaluates to the first space-delimited word in the name. Applied to a list, equivalent to <list>:first:elide.
empty	Applied to a list, evaluates to a boolean, which is `TRUE` if the list is empty.
false	Applied to a boolean, evaluates to a boolean, which is the logical negation of its input.
file	Applied to a controlled unit name, evaluates to a string that contains the path of the file associated with (providing persistent storage for) that controlled unit. Applied to a list of controlled unit names, evaluates to a string that contains a list of paths using the preceding rule for each controlled unit name in the input list.
first	Applied to an empty list, evaluates to `NULL`. Applied to a non-empty list, evaluates to a string that contains the first element of the list.
headerfile	Applied to a unit or item name or a list of unit or item names, does one of the following: ▪ Evaluates to a string that contains the name of the item's associated header file. ▪ Evaluates to a string that contains a list of corresponding header file names.

Table 4 Menu File Modifiers (continued)

Modifier	Description
home_unit	Applied to a model component name, evaluates to a string that contains the name of the controlled unit in which the item is located.
multiple	Applied to a list of unit, item, or file names, evaluates to a boolean, which is `TRUE` if list has more than one element.
not	A synonym (and preferred method) for false.
sourcefile	Applied to a unit or item name or to a list of unit or item names, does one of the following: ▪ Evaluates to a string, that contains the name of the item's associated sourcefile. ▪ Evaluates to a string that contains a list of corresponding sourcefile names.
unary	Applied to a list, evaluates to a boolean, which is `TRUE` if the input list has exactly one element.
writeable	Applied to a path resolving to a file, evaluates to a boolean, which is `TRUE` if the file is writable. Applied to a controlled unit name, evaluates to a boolean, which is `TRUE` if the controlled unit is writable.

Syntax Rules for Rose Menu File Entries

Follow these rules when specifying menu text:

- When a text string contains embedded spaces, enclose the string in double quotation marks.

  For example, `"Run Script"`

- When a text string has no embedded spaces (a single word, for example), enter the string without any quotation marks.

  For example, `Validate`

- When a text string that is not enclosed in quotes includes a special character, the special character could be misinterpreted as a variable. For this reason, you must precede any special characters (such as ^, “, or %) with an escape character. The escape character for all special characters is ^.

  Examples:

  - `Option Calculate^%`

    Creates a menu option whose text reads **Calculate %**.

- `exec Notepad ^""c:\my files\file.txt"^"`

  Creates a menu action that executes the following command line: **notepad "c:\my files\file.txt"**.

  Note the escape character followed by an additional set of quotation marks. One set of quotation marks is necessary because there is a space in `my files`. The second set, each of which is preceded by the ^ escape character, causes the actual command line to include the quotation marks as part of the command.

- To create a mnemonic for the menu, add an ampersand (&) before the menu text.

  For example, `"&Run Script"`

  Allows users to execute the menu item by pressing CTRL+R.

- Menu text can include variables and modifiers.

  For example, `Option "Validate "%model`

  Creates a menu option with the text **Validate MyCurrentModel** (assuming the current model is called MyCurrentModel).

  See *Menu File Variables and Modifiers* on page 11 for more information.

Adding Scripts to a Rose Menu

If you create a Rose script that you will use over and over again, you may want to add it to a Rose menu. For example, if you write a script to create a particular report based on the contents of a model, you will probably run that script periodically.

To add such scripts to a Rose menu:

1. Open the Rose Menu file, or create a new one to use in its place.
2. Edit the Path Map so that it includes a virtual script path. (See *Adding or Editing the Virtual Path for Scripts* on page 16.)
3. Modify the Rose menu file to add the script under the appropriate menu, being careful to follow all of the menu file syntax rules. To do this:
   - In the menu file, locate the menu specification that corresponds to the Rose menu to which you want to add the script. Each menu specification is comprised of the Menu keyword followed by the name of a Rose menu. For example, the **Tools** menu specification begins with **Menu Tools**.
   - Within the appropriate menu specification, add a menu option that specifies the text of the menu command that will run the script (for example, **Run Conversion Wizard**).

- Enter a RoseScript menu action to cause the script to execute when a user selects the menu command.

4 Save the updated menu file.

Adding or Editing the Virtual Path for Scripts

When you edit the Rose menu file to include script commands, you must include one of the following:

- The fully qualified name of the script file to execute
- The virtual path that maps to the actual path

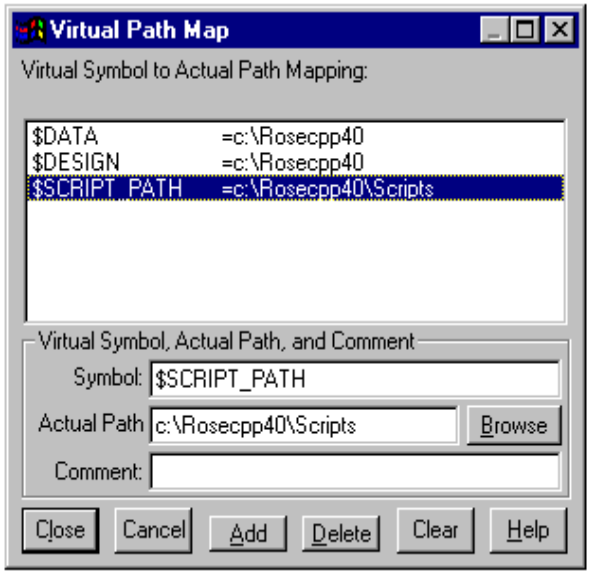
Defining a virtual path for scripts simplifies the process of editing the menu file by allowing you to specify the symbolic virtual path name instead of the complete file path.

To add or edit a virtual path for scripts:

1 Start Rose.

2 Click **File > Edit Path Map** to display the **Virtual Path Map** dialog box.

Figure 3 Adding Virtual Path for Scripts



3 Check for the $SCRIPT_PATH virtual symbol and do one of the following:

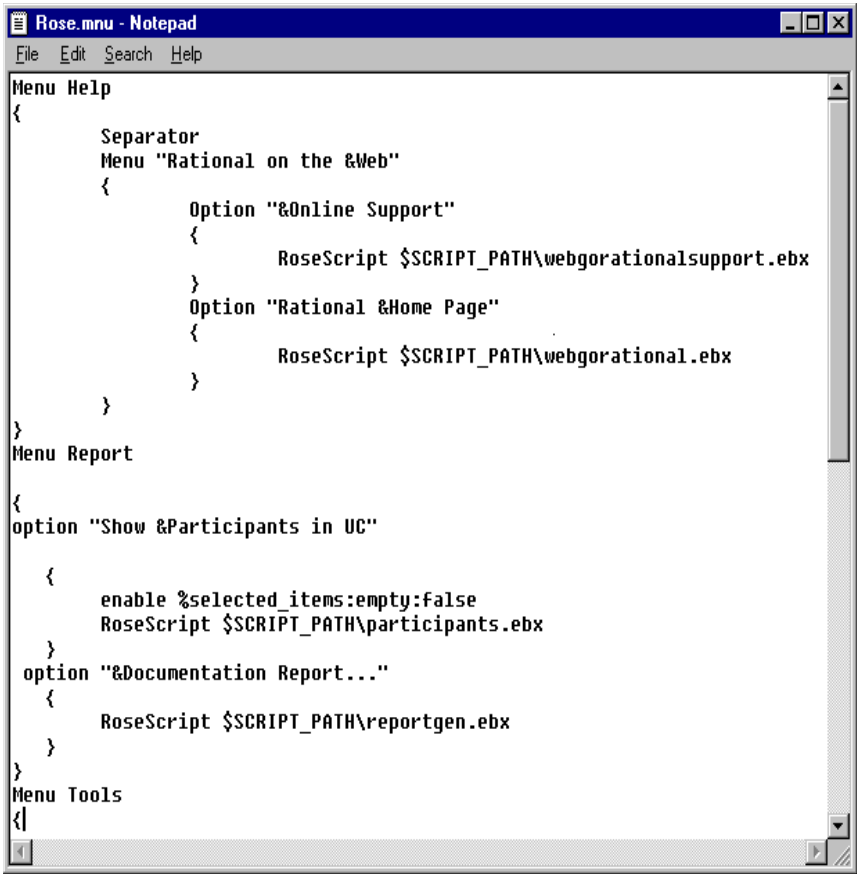
- If the symbol exists, select it in the dialog box to display its current mapping information in the lower portion of the dialog box.

- If the symbol does not exist, enter it in the **Symbol** field in the lower portion of the dialog box.
- Enter the actual path to your Rose scripts, or use the **Browse** button to locate and select the path. (Normally these scripts reside in a Scripts subdirectory of the Rose installation directory.)
- When you make changes in the dialog box, the **Close** button becomes an **OK** button. Click **OK** to save your changes and exit the **Virtual Path Map** dialog box.

Sample Rose Menu File

The following figure shows a portion of a Rose menu file.

Figure 4 Sample Rose Menu File



Note the following entries as you examine the menu specifications that comprise this file:

- Separator Entry

  A separator entry causes a separator line to appear between the menu item above the keyword and the menu item below the keyword. In this case, a separator line will appear above the **Rational on the Web** submenu.

- Menu Entry

  A Rose menu entry consists of the Menu keyword, followed by the name of the Rose menu being extended.

  This menu file extends the **Help** menu, the **Reports** menu, and the **Tools** menu (only partially in view).

- Submenu Entry

  A submenu is a second level menu that appears under a menu. A submenu entry looks just like a menu entry, with two exceptions:

  - It appears within a Rose menu specification. (In this case, it is part of the Rose **Help** menu specification.)
  - The Menu keyword is followed by the submenu title. (Notice that when the submenu title has embedded spaces, it is enclosed in quotation marks.)

- Option Entry

  Menu option entries define menu commands that you add to a menu. They begin with the Option keyword and are followed by the option title.

  This file adds the following commands to the specified menus:

  - Online Support (**Rational on the Web** submenu of the **Help** menu)
  - Rational Home Page (**Rational on the Web** submenu of the **Help** menu)
  - Show Participants in UC (**Report** menu)
  - Documentation Report (**Report** menu)

- Menu Action

  Menu actions tell Rose what to do when the menu item is selected. Each of the options in this file executes a Rose Script. The Menu Actions topic describes all of the available actions.

- Menu Argument

  Menu arguments can be included to enable or disable a menu item under various circumstances. The arguments begin with either an Enable or Disable keyword, followed by variables and modifiers which define the circumstances.

In this case, the **Show Participants in UC** menu item will be enabled only if at least one item is selected in the current diagram (that is, the statement, “The list of selected items is empty” is false).

- Braces

  Every menu specification entry must begin with a left brace ({) and end with a right brace (}). The nested braces allow you to define the hierarchy of a menu specification from the Rose menu at the high end to a submenu option at the low end.

Customizing Rose Shortcut Menus

When you or the user of your add-in to Rose right-clicks in Rose, a shortcut menu appears. The commands displayed on the shortcut menu are determined by where you or your add-in user clicks the mouse and what items are selected in the diagram or browser. You can take advantage of this feature in your add-in’s functionality so that your add-in user sees your shortcut menu items when they right-click. If your add-in has features that you want to include on a shortcut menu, the shortcut menu Help topics explain how to add items to the Rose shortcut menu by using the Rose Extensibility Interface (REI).

Benefits

The REI exposure of Rose’s shortcut menu interface provides the following benefits:

- Quicker access to your add-in’s features for your customers
- Control of when the menu item appears on the shortcut menu
  - Default (any time the user selects multiple different items, such as classes and packages, or has nothing selected)
  - Diagram
  - Package
  - UseCase
  - Class
  - Attribute
  - Operation
  - Component
  - Role
  - Properties

    - Model
    - DeploymentUnit
    - ExternalDoc
- Control of the state in which your menu item appears on the shortcut menu (enabled, disabled, checked, unchecked)
- Control of the order (but not position) of multiple menu items on the shortcut menu
- Ability to add submenus to shortcut menu items
- Ability to add separator lines to the shortcut menu and submenus
- Ability to create one shortcut menu item that works for items selected in the browser as well as in a diagram (you do not have to create one menu item for items selected in the browser and another menu item for items selected in the diagram)

Limitations

The position on the shortcut menu where your menu item appears is controlled by Rose. If you have more than one item on the shortcut menu, however, you can control the order in which those items appear by adding the items (using the AddContextMenuItem method) in the order in which you want the menu items to appear.

Key Terms and Concepts

Language-Dependent

Rose model elements are language-dependent if they can be associated with a specific language add-in, especially for code generation. These language-dependent model elements are:

- Associations
- Attributes
- Classes
- Components
- Operations
- Roles

Language-Neutral

Rose model elements are language-neutral if they are not associated with a specific language. They are not generated into code (although model elements within them can be generated into code). These language-neutral model elements are:

- Activities
- Decisions
- DeploymentUnits
- Diagrams
- ExternalDocs
- Models
- Packages
- Properties
- States
- Subsystems
- Swimlanes
- Synchronizations
- Transitions
- UseCases

Even though these model elements are language-neutral, a language add-in can work with them (except for DeploymentUnits, ExternalDocs, Models, and Properties) in the following ways. The language add-in can:

- Add shortcut menu items.
- Get the OnContextMenuItem event for them, as long as the language add-in is set as the default language.

In other words, if your language add-in is the default language, when a user right-clicks on any of the above language-neutral model elements (except for DeploymentUnits, ExternalDocs, Models, and Properties), the user sees your language add-in's shortcut menu items for these model elements. If your language add-in is not the default language and the user right-clicks on one of the language-neutral model elements (except for DeploymentUnits, ExternalDocs, Models, and Properties), the user does not see your shortcut menu items for these model elements.

Language Add-In

A language add-in is an add-in whose Rose Registry setting for "LanguageAddIn" is set to Yes.

Non-Language Add-In

A non-language add-in is an add-in whose Rose Registry setting for "LanguageAddIn" is set to No.

Behind the Scenes of Shortcut Menus

When Rose is started, it issues the OnActivate event and gets shortcut menu items (ContextMenuItem) from each add-in.

When you or the user of your add-in right-clicks, Rose sends the OnEnableContextMenuItems event to the appropriate add-ins to get the applicable menu states for the add-in's ContextMenuItems. Rose then formats and displays the shortcut menu with appropriate add-in menu items depending on:

- Where the user right-clicked.
- What items are selected (such as class and package).
- For what context the add-in's shortcut menu items are defined (ContextMenuItemType).

See *How Rose Formats and Displays Shortcut Menu Items* on page 22.

When the user selects a menu item from the shortcut menu, Rose sends the OnSelectedContextMenuItem to the appropriate add-ins. It is then up to the add-in to map the event and arguments to one of its methods.

The add-in then runs the method that corresponds to the selected shortcut menu item.

How Rose Formats and Displays Shortcut Menu Items

The methodology for determining which shortcut menu items appear and when they appear makes the add-in a smart, seamless part of Rose. Rose only displays your shortcut menu when it is appropriate to do so. For example, Rose displays your class shortcut items for a particular class if the following conditions are all true:

- Your add-in is a language add-in (defined by the registry setting)
- Your user creates a class with your language
- You have created the appropriate shortcut menu items (ContextMenuItems)

However, if your user creates a class with a different language add-in, Rose does not display your class shortcut menu items. (A menu item for C++ classes might not make sense for a Visual Basic class.) Because of the flexibility that Rose gives you through the REI to create shortcut menu items, it is also a complex concept. A shortcut menu item might not appear when expected. It is, therefore, important to understand the scenarios (explained later in this chapter) for the complete explanation of what is displayed and when it is displayed.

Shortcut menu items created by a non-language add-in are always displayed on the appropriate menu. Those items created by a language add-in are displayed when the selected items have that language assignment. Shortcut menu items created by a language add-in are also displayed when language-neutral items are selected and that language is the **Default Language**. Whether Rose displays a particular shortcut menu item is dependent on the following considerations:

- The **Default Language** setting (click **Tools > Options > Notation**)
- Whether the selected items are the same type
- Whether the selected items are language-dependent or language-neutral
- Whether the selected items are associated with a particular language add-in

See *Shortcut Menu Scenarios* on page 23 for examples of how these issues affect shortcut menu items.

Shortcut Menu Scenarios

The following table describes all the possible scenarios for displaying shortcut menu items.

Table 5 Displaying Shortcut Menu Items

Description	Example Selected Items	Displayed Shortcut Menu Items
Same language-dependent types selected	Class 1 (Language A) (Any Default Language)	Language add-in A's rsClass shortcut menu items. All non-language add-in's rsClass shortcut menu items.
	Class 1 (Language A) Class 2 (Language A) (Any Default Language)	Language add-in A's rsClass shortcut menu items. All non-language Add-in's rsClass shortcut menu items.
	Class 1 (Language A) Class 2 (Language B) (Any Default Language)	Language add-in A's rsClass shortcut menu items. Language add-in B's rsClass shortcut menu items. All non-language Add-in's rsClass shortcut menu items.
	Attribute 1 (Language A) Attribute 2 (Language B) (Any Default Language)	Language add-in A's rsAttribute shortcut menu items. Language add-in B's rsAttribute shortcut menu items. All non-language add-in's rsAttribute shortcut menu items.

Table 5 Displaying Shortcut Menu Items (continued)

Description	Example Selected Items	Displayed Shortcut Menu Items
Different language-dependent types selected	Class 1 (Language A) Class 2 (Language B) Attribute 3 (Language A) Attribute 4 (Language C) (Any Default Language)	Language add-in A's rsDefault shortcut menu items. Language add-in B's rsDefault shortcut menu items. Language add-in C's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
	Operation 1 (Language A) Operation 2 (Language B) Role 3 (Language A) (Any Default Language)	Language add-in A's rsDefault shortcut menu items. Language add-in B's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
Nothing selected	(Default Language set to Language A)	Language add-in A's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
	(Default Language set to Language B)	Language add-in B's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
Same language-neutral type selected	Diagram 1 (Created with any language) (Default Language set to Language A)	Language add-in A's rsDiagram shortcut menu items. All non-language add-in's rsDiagram shortcut menu items.
	Diagram 1 (Created with any language) (Default Language set to Language B)	Language add-in B's rsDiagram shortcut menu items. All non-language add-in's rsDiagram shortcut menu items.
	Diagram 1 (Created with any language) Diagram 2 (Created with any language) (Default Language set to Language A)	Language add-in A's rsDiagram shortcut menu items. All non-language add-in's rsDiagram shortcut menu items.

Table 5 Displaying Shortcut Menu Items (continued)

Description	Example Selected Items	Displayed Shortcut Menu Items
Same language-neutral type selected, continued	Diagram 1 (Created with any language) Diagram 2 (Created with any language) (Default Language set to Language B)	Language add-in B's rsDiagram shortcut menu items. All non-language add-in's rsDiagram shortcut menu items.
Different language-neutral types selected	Diagram 1 (Created with any language) Package 2 (Created with any language) Package 3 (Created with any language) (Default Language set to Language A)	Language add-in A's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
	Subsystem 1 (Created with any language) UseCase 2 (Created with any language) (Default Language set to Language B)	Language add-in B's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
Combination of language-dependent and language-neutral types selected	Class 1 (Language A) Package 2 (Created with any language) (Default Language set to Language B)	Language add-in A's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.
	Class 1 (Language A) Class 2 (Language B) Package 3 (Created with any language) Package 4 (Created with any language) (Default Language set to Language C)	Language add-in A's rsDefault shortcut menu items. Language add-in B's rsDefault shortcut menu items. All non-language add-in's rsDefault shortcut menu items.

Note: If you want a shortcut menu item to appear on more than one menu (for example, for classes and the default), you must create a separate ContextMenuItem for each item type (for example, one for rsClass and one for rsDefault). For examples, see *Sample Shortcut Menu Implementation Code* on page 28 and *Sample Rose Script Shortcut Menu Code* on page 30.

For more information on rsClass, rsAttribute, rsDefault, or rsDiagram see *ContextMenuItemType Enumeration* in the online Help.

Shortcut Menu Design Considerations

To keep the shortcut menu from becoming too cluttered with many different add-in menu options, try to keep menu items on the main shortcut menu to a minimum. Use submenus as much as possible. However, put all the important menu options on the main shortcut menu. Put less important menu options on a submenu under a generic main shortcut menu option.

Generate Code and **Browse Code** are not standard Rose shortcut menu options. Each language add-in is responsible for creating and manipulating these options according to their needs. This gives you greater control and flexibility with these features. When creating menu items, make the caption specific to your language (for example, Generate C++ Code, Generate Visual Basic Code). This reduces confusion since the user of your add-in could be using more than one language add-in in Rose. Place **Generate Code** and **Browse Code** at the top level of the shortcut menu instead of in a submenu.

You could also place shortcut menu items that open a custom specification sheet at the top level of the shortcut menu. However, if your add-in supports the OnPropertySpecOpen event, do not add a custom specification menu item because it would be redundant. This is due to the fact that when Rose detects that the OnPropertySpecOpen event is supported for an item, Rose adds the **Open Standard Specification** shortcut menu item (which displays the standard Rose specification) immediately after the **Open Specification** shortcut menu item (which, in this context, displays the add-in's custom specification sheet).

Procedure

To customize the Rose shortcut menu:

1. In order to use this feature of the REI, you must register your product in the Rose Add-In Manager.
2. Determine the following:
   - What are your menu items?
   - Through what states will each menu item go?

- Where should each menu item appear (main shortcut menu or a submenu)?
- In what order should your menu items appear on the shortcut menu or submenu?
- In which contexts should your menu items appear (for example, rsDefault, rsClass, and rsPackage).
- What circumstances will change menu states for each menu item?
- Which access keys, if any, should you assign to each menu option?
- Are there any other considerations that are specific to your implementation?

3. Create the prototyped event methods, OnActivate, OnEnableContextMenuItems, and OnSelectedContextMenuItem customizing for your specific needs.
4. Create ContextMenuItem objects for each menu item by using the AddContextMenuItem method for each menu item. Use AddContextMenuItem in the order in which you want the menu items displayed on the shortcut menu.
5. Create your specific methods to support each ContextMenuItem (shortcut menu item) that maps to a specific function of your add-in. If the method already exists, update it as needed to take advantage of the Rose shortcut menu.
6. Create and incorporate menu state changes as needed for your add-in. Use the MenuState property of the ContextMenuItem to change menu states.
7. Determine if there are any additional steps necessary for your specific implementation and perform those steps.

Adding Menu Items to the Shortcut Menu

To create and add menu items to the shortcut menu, use the AddContextMenuItem Method.

Note: An add-in should add context menu items when it gets the OnActivate event.

Working with Shortcut Menu Items

When the user activates the shortcut menu with items selected in the browser or a diagram, Rose sends the OnEnableContextMenuItems event to the specified language add-in. The language add-in can then call GetSelectedItems at the model level to get all selected items, regardless of whether the user selected the items in the browser or in a diagram.

Working with the Shortcut Menu Item Collection

To work with a subset or the set of all shortcut menu items, use the GetContextMenuItems Method.

To enable, disable, check, or uncheck shortcut menu items:

1. Iterate through the collection of ContextMenuItem objects by using the GetAt method.
2. Set the MenuState property accordingly.

Editing Shortcut Menu Items

To change the properties of the shortcut menu item, see the ContextMenuItem class properties and methods.

Changing the State of a Shortcut Menu Item

To enable, disable, check, or uncheck a particular shortcut menu item, change the ContextMenuItem's MenuState property.

Sample Shortcut Menu Implementation Code

The following are sample pieces of code that you might use to add your menu items to Rose's shortcut menu.

```
'Customize OnActivate from the prototype
Sub OnActivate (LPDISPATCH pRoseApp)
   . . .
   'Create all shortcut menu items
   Set myNewMenuItem1 =
      myAddIn.AddContextMenuItem (rsDefault, "Separator", "")
   Set myNewMenuItem2 = myAddIn.AddContextMenuItem (rsDefault,
      "Submenu &Main Add-In Menu Caption", "")
   Set myNewMenuItem3 = myAddIn.AddContextMenuItem (rsDefault,
      "&Caption 1", "internalName1")
   Set myNewMenuItem4 = myAddIn.AddContextMenuItem (rsDefault,
      "C&aption 2", "internalName2")
   Set myNewMenuItem5 = myAddIn.AddContextMenuItem (rsDefault,
      "endsubmenu", "")
   Set myNewMenuItem6 = myAddIn.AddContextMenuItem (rsDefault,
      "Separator", "")
. . .
End Sub 'OnActivate event
```

```
. . .
'Set initial state of each selectable shortcut menu item
myNewMenuItem2.MenuState = ENABLED
myNewMenuItem3.MenuState = ENABLED
myNewMenuItem4.MenuState = DISABLED
. . .

'Customize OnEnableContextMenuItems from the prototype
Function OnEnableContextMenuItems (LPDISPATCH pRoseApp, VT_I2
itemType) As Boolean
   . . .
End Function 'OnEnableContextMenuItems event

. . .

'Create each routine that corresponds to a selectable shortcut
'menu item
Sub DoMenuOption1 (argument1, argument 2, …)
   . . .
End Sub 'DoMenuOption1 subroutine

Function DoMenuOption2 (argument1, argument2, º) As returnValue2
   . . .
End Function 'DoMenuOption2 function

. . .

'Customize OnSelectedContextMenuItem from the prototype to map
'selectable shortcut menu items to functionality in this
'    add-in.
Function OnSelectedContextMenuItem (LPDISPATCH pRoseApp, BSTR
internalName) As Boolean
   Select Case internalName
      Case internalName1
      DoMenuOption1 (argument1, argument2, …)
      Case internalName2
```

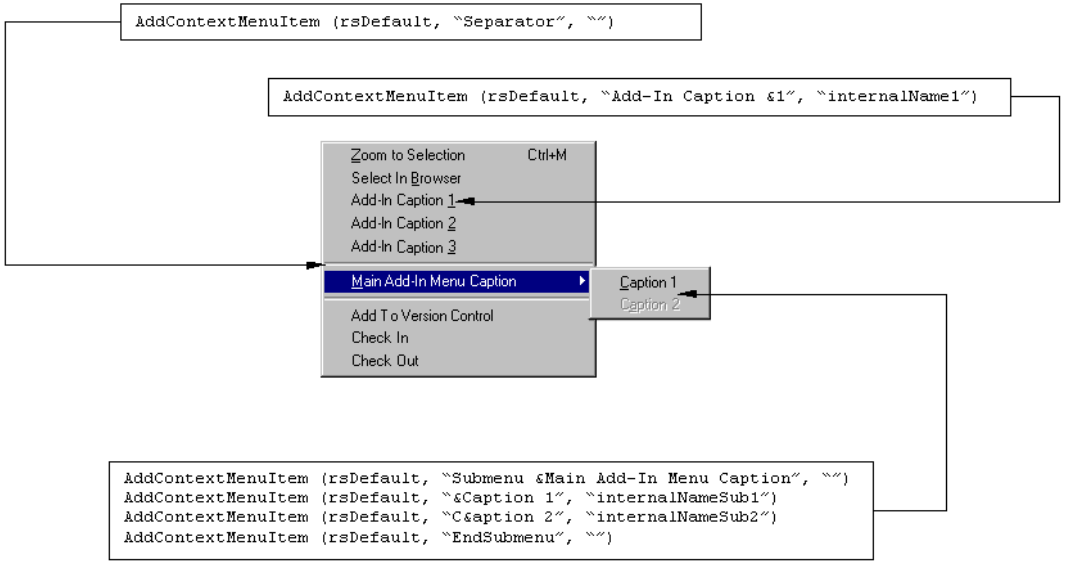
```
      x = DoMenuOption2 (argument1, argument2, …)
   End Select 'internalName
   . . .
End Function 'OnSelectedContextMenuItem event

'Main program functionality
Sub Main
   . . .
End Sub 'Main Program
```

Sample Rose Script Shortcut Menu Code

The sample RoseScript code below produced the shortcut menu in Figure 5 on page 30.

Figure 5 Sample Code for Shortcut Menus




```
'Subroutines to which the selectable shortcut menu items map
Sub internalName1

End Sub

Sub internalName2
```

```
End Sub

Sub internalName3

End Sub

Sub internalNameSub1

End Sub

Sub internalNameSub2

End Sub

Sub internalNameClass1

End Sub

Sub internalNameClass2

End Sub

Sub internalNameClass3

End Sub

Sub internalNameClassSub1

End Sub

Sub internalNameClassSub2

End Sub

Sub Main
```

```
'Create a sample shortcut menu
Dim myAddIn As RoseAddIn
Dim myMenuItem As ContextMenuItem
Dim myMenuItem2 As ContextMenuItem
Dim myMenus As ContextMenuItemCollection
Dim menuCount As Integer
Dim i As Integer
Dim classFound As Boolean
Dim myItems As ItemCollection
Dim itemCount As Integer
Dim anItem As RoseItem
Dim myModel As Model

'ContextMenuItemType enumeration
Const rsDefault As Integer = 0
Const rsClass As Integer = 4

'MenuState enumeration
Const rsDisabled As Integer = 0
Const rsEnabled As Integer = 1

Set myAddIn = ... 'Get the add-in to which you want to add
                  'shortcut menu items.

'Create shortcut menu items for rsDefault
Set myMenuItem = myAddIn.AddContextMenuItem(rsDefault, "Add-In
Caption &1", "internalName1")
Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault, "Add-In
Caption &2", "internalName2")
Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault, "Add-In
Caption &3", "internalName3")
Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault,
"Separator", "")
```

```
Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault, "Submenu
&Main Add-In Menu Caption", "")

Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault, "&Caption
1", "internalNameSub1")

Set myMenuItem2 = myAddIn.AddContextMenuItem (rsDefault, "C&aption
2", "internalNameSub2")

Set myMenuItem2.MenuState = rsDisabled

Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault,
"endsubmenu", "")

Set myMenuItem = myAddIn.AddContextMenuItem (rsDefault,
"Separator", "")


‘Create exact same shortcut menu items for rsClass

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Add-In
Caption &1", "internalNameClass1")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Add-In
Caption &2", "internalNameClass2")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Add-In
Caption &3", "internalNameClass3")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Separator",
"")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Submenu &Main
Add-In Menu Caption", "")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "&Caption 1",
"internalNameClassSub1")

Set myMenuItem2 = myAddIn.AddContextMenuItem (rsClass, "C&aption 2",
"internalNameClassSub2")

Set myMenuItem2.MenuState = rsDisabled

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "endsubmenu",
"")

Set myMenuItem = myAddIn.AddContextMenuItem (rsClass, "Separator",
"")



‘Check to see if the user has selected only Class items. If

‘so, enable the disabled shortcut menu option (Caption 2
```

```
    'on the submenu).
    classFound = True
    Set myItems = RoseApp.CurrentModel.GetSelectedItems ()
    itemCount = myItems.Count
    For i = 1 To itemCount
       Set anItem = myItems.GetAt (i)
       If anItem.Stereotype <> "Class" Then
       classFound = False
       End If
    Next i

    If classFound = True Then
       myMenuItem2.MenuState = rsEnabled
    End If

End Sub
```

Using the REI to Work with Rational Rose

3

Contents

This chapter is organized as follows:

- *Introduction* on page 35
- *Getting the Rose Application Object* on page 36
- *Associating Files and URLs with Classes* on page 37
- *Managing Default Properties* on page 37
- *Adding a Property to a Set* on page 39
- *Creating a New Property* on page 40
- *Deleting Model Properties* on page 41
- *Getting Model Properties* on page 41
- *Setting Model Properties* on page 41
- *Creating a New Property Set* on page 43
- *Cloning a Property Set* on page 44
- *Deleting a Property Set* on page 45
- *Getting and Setting the Current Property Set* on page 46
- *Creating a User-Defined Property Type* on page 47
- *Creating a New Tool* on page 48
- *Placing Classes in Categories* on page 48
- *Using Type Libraries for Rose Automation* on page 49
- *Working with Controllable Units* on page 49
- *Working with Rose Diagrams* on page 50
- *Getting an Element from a Collection* on page 51

Introduction

This chapter explains how to use the Rational Rose Extensibility Interface (REI) to accomplish many tasks that you would otherwise perform manually in the Rose user interface.

This information is meant to orient you and to provide examples that you can use as starting points in your work with the REI.

The information in this chapter is not exhaustive. You should refer to the Extensibility online Help for complete descriptions of all of the REI classes, properties, and methods. As you familiarize yourself with these, you will be able to realize the full capabilities that the REI makes available to you.

Getting the Rose Application Object

Whether you are using Rose Script or Rose Automation, you must get the Rose Application object in order to control the Rose application.

Using Rose Script

All Rose Script programs have a global object called RoseApp, which has a property called CurrentModel. You must use RoseApp.CurrentModel to initialize the global Rose object and subsequently open, control, save, or close a Rose model from within a script.

The following sample code shows how to get the Rose Application object in a Rose Scripting context:

```
Sub GenerateCode (theModel As Model)
   ' This generates code
End Sub

Sub Main
   GenerateCode RoseApp.CurrentModel
End Sub
```

Using Rose Automation

To use Rose as an automation server, you must initialize an instance of a Rose application object. You do this by calling either CreateObject or GetObject (or their equivalents) from within the application you are using as the OLE controller. These calls return the OLE Object which implements Rose API's application object.

Refer to the documentation for the application you are using as OLE controller for details on calling OLE automation objects.

The following sample code shows how to get the Rose application object in a Rose Automation context:

```
Sub GenerateCode (theModel As Object)
   ' This generates code
End Sub
```

```
Sub Main
   Dim RoseApp As Object
   Set RoseApp = CreateObject ("Rose.Application")
   GenerateCode RoseApp.CurrentModel
End Sub
```

Associating Files and URLs with Classes

Because Class objects inherit properties from RoseItem, you can define a set of external documents for any class. Each external document has either a Path property or a URL property.

- The Path property specifies a path to the file that contains the external document.
- The URL property specifies a Universal Resource Locator (URL) of a corresponding internet document.

Note: See the Extensibility online Help for syntax and other information.

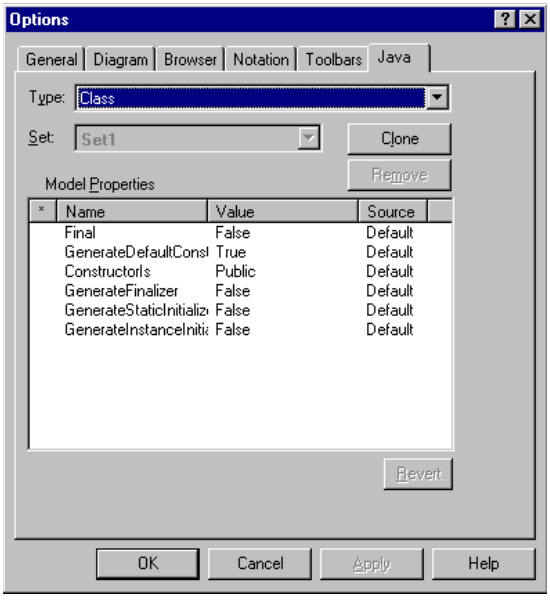
Managing Default Properties

In the Rose user interface environment, you manage a model's properties by using the specification editor.

To access the specification editor, click **Tools > Model Properties > Edit**.

You then select the appropriate tool tab, element type, and property set to edit. For example, in Figure 6 on page 38, the tool is **Java**, the model element type is **Class**, and the property set is **Set1**.

Figure 6 Property Specification Editor



From this point on, you can use the specification editor to edit individual properties, as well as clone (copy) and edit property sets. However, you cannot create new tools (tabs), new default property sets, or property types. For these capabilities, you must use the Rose Extensibility Interface.

For more information on editing default properties and sets in the Rose user interface, see the online Help for information on specifications, or refer to the *Using Rose* manual.

The next sections of this chapter explain how to work with properties and property sets in the extensibility environment.

In the Extensibility Interface, the DefaultModelProperties object manages the default model properties for the current model, and is itself a property of the model (expressed as RoseApp.CurrentModel.DefaultProperties). For this reason, default properties are applied to the current model only. When you create default properties they are applied to and saved for the current model, but are not available to any new models you create.

To apply new properties to another model, re-run the script that creates the properties, specifying the new model as the current model.

Adding a Property to a Set

How To

To add a property to a property set, define a subroutine that uses the DefaultModelProperties.AddDefaultProperty method. You will notice that this method requires you to pass 6 parameters:

- Class Name
- Tool Name
- Set Name
- Name of the New Property
- Property Type
- Value of the New Property

Example

```
Sub AddDefaultProperties (theModel As Model)
   Dim DefaultProps As DefaultModelProperties
   Set DefaultProps = theModel.DefaultProperties
   myClass$ = theModel.RootCategory.GetPropertyClassName ()
   b = DefaultProps.AddDefaultProperty (myClass$,
      "ThisTool", "Set1", "StringProperty", "String", "")
   b = DefaultProps.AddDefaultProperty (myClass$,
      myTool$, "Set1", "IntegerProperty", "Integer", "0")
   b = DefaultProps.AddDefaultProperty (myClass$,
      myTool$, "Set1", "FloatProperty", "Float", "0")
   b = DefaultProps.AddDefaultProperty (myClass$,
      myTool$, "Set1", "CharProperty", "Char", " ")
   b = DefaultProps.AddDefaultProperty (myClass$,
      myTool$, "Set1", "BooleanProperty", "Boolean", "True")
End Sub
```

Notes on the Example

1. When you specify the Class Name parameter, you must specify the internal name of the model element. There are two ways to obtain this information:
   - If properties are already defined for this element, the internal name appears in the specification dialog box in the Rose user interface. Simply check the specification editor and use the **Type** drop-down list to find the appropriate class name.
   - Use the Element.GetPropertyClassName method. This is the method used in the sample script. This example retrieves the internal name and returns it in myClass$, which is then passed as the class name parameter.
2. If the tool you specify does not exist, a new tool will be created. This is actually the only way to add a new tool to a model.
3. This example adds a property of each of the predefined property types (string, integer, float, char, boolean), with the exception of the enumeration type. You use the enumerated type to create your own property types and add enumerated properties to a set. See *Creating a User-Defined Property Type* on page 47 for instructions and an example.

Creating a New Property

How To

To create a new property that is not based on an existing property, use the Element.CreateProperty method. However, if you simply want to set an existing property to a different current value, you should use Element.InheritProperty or Element.OverrideProperty instead.

Example

```
' Property creation:
b = theModel.RootCategory.CreateProperty (myTool,
   "Saved", "True", "Boolean")
b = theModel.RootCategory.InheritProperty (myTool, "Saved")
```

Notes on the Example

1. The CreateProperty call in the example creates a new property called Saved. It applies to the tool MyTool, its value is `TRUE`, and its type is Boolean.
2. The InheritProperty call in the example deletes the property just created. Because there is no default value to which such a property can return, InheritProperty effectively deletes it from the model.
3. For more information, see *Setting Model Properties Using InheritProperty* on page 43, *Setting Model Properties Using OverrideProperty* on page 42, and *Deleting Model Properties* on page 41.

Deleting Model Properties

If you are deleting a property that belongs to a property set, you can use the DefaultModelProperties.DeleteDefaultProperty method to delete the property from a model.

However, if you created a property using the Element.CreateProperty method, that property is not part of a property set. To delete such a property, use the Element.InheritProperty method.

Getting Model Properties

The Element class provides two methods for retrieving information about model properties:

- To get the current value for a model property, whether inherited or overridden, use the Element.GetPropertyValue method. This method returns the value as a string.
- To retrieve the property object itself, use the Element.FindProperty method.

Setting Model Properties

There are several ways to set model properties using the Extensibility Interface:

- Use the Element.OverrideProperty method to change only the value of a property, and keep all other aspects of the property definition intact.
- Use the Element.InheritProperty method to return a previously overridden property to its original value.

- Use the Element.CreateProperty or the DefaultModelProperties.AddDefaultProperty method to define a new property that is not based on an existing property.

For more information, see *Creating a New Property* on page 40. For more information on creating new properties that are based on an existing property set, see *Cloning a Property Set* on page 44.

Setting Model Properties Using OverrideProperty

How To

The Element.OverrideProperty method allows you to use the default property definition and simply change its current value. Alternately, you could create a brand new property by calling the Element.CreateProperty method, but that would require you to specify the complete property definition, not just the new value.

If the property you specify does not exist in the model's default set, a new property is created for the specified object only. This new property is created as a string property.

Example

```
Sub OverrideRadioProps (theCategory As Category)
   b = theCategory.OverrideProperty (myTool$, "StringProperty", "This
   string is overridden")
   b = theCategory.OverrideProperty (myTool$, "IntegerProperty", "1")
   b = theCategory.OverrideProperty (myTool$, "FloatProperty",
   "111.1")
   b = theCategory.OverrideProperty (myTool$, "EnumeratedProperty",
   "Value2")
End Sub
```

Notes on the Example

1 Each of the 4 lines of the sample subroutine changes the current value of a specific property as follows:

- The property called StringProperty now has a value of "This string is overridden".
- The property called IntegerProperty now has a value of 1.
- The property called FloatProperty now has a value of 111.1.
- The property called EnumeratedProperty now has a value of "Value2".

2 Everything except for current value (tool name, class name, set, property name, and property type) remains the same for the properties.

Setting Model Properties Using InheritProperty

How To

Use the Element.InheritProperty method to reset an overridden property to its original value.

You can also use this method to delete a property that you created using the Element.CreateProperty method. Because there is no default value to which such a property can return, InheritProperty effectively deletes it from the model.

Example

```
Sub InheritRadioProps (theCategory As Category)
   b = theCategory.InheritProperty (myTool$, "StringProperty")
   b = theCategory.InheritProperty (myTool$, "IntegerProperty")
   b = theCategory.InheritProperty (myTool$, "FloatProperty")
   b = theCategory.InheritProperty (myTool$,
      "EnumeratedProperty")
End Sub
```

Notes on the Example

Each of the 4 lines of the sample subroutine returns the current value of the specified property to its original value.

Creating a New Property Set

To create a new property set that is not based on an existing property set, use the DefaultModelProperties.CreateDefaultPropertySet method.

Cloning a Property Set

How To

Cloning allows you to create a copy of an existing property set for the purpose of creating another property set. This is the easiest way to create a new property set, and is particularly useful for creating multiple sets of the same properties, but with different values specified for some or all of the properties.

To clone a property set in a model, use the DefaultModelProperties.CloneDefaultPropertySet method.

Example

```
Sub CloneDefaultProperties (theModel As Model)
   Dim DefaultProps As DefaultModelProperties
   Set DefaultProps = theModel.DefaultProperties
   AddDefaultProperties theModel
   myClass$ = theModel.RootCategory.GetPropertyClassName ()
   b = DefaultProps.CloneDefaultPropertySet (myClass$, myTool$,
   "default", "SecondSet")
   b = DefaultProps.CloneDefaultPropertySet (myClass$, myTool$,
   "default", "ThirdSet")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$,
   "SecondSet", "StringProperty", "String", "Unique to SecondSet")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$,
   "SecondSet", "IntegerProperty", "Integer", "11")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$,
   "SecondSet", "FloatProperty", "Float", "89.9000")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$,
   "SecondSet", "EnumeratedProperty", "EnumerationDefinition",
   "Value2")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$, "ThirdSet",
   "StringProperty", "String", "Unique to ThirdSet")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$, "ThirdSet",
   "IntegerProperty", "Integer", "20")
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$, "ThirdSet",
   "FloatProperty", "Float", "90.9000")
```

```
   b = DefaultProps.AddDefaultProperty (myClass$, myTool$, "ThirdSet",
   "EnumeratedProperty", "EnumerationDefinition", "Value3")
End Sub
```

Notes on the Example

1. This example clones an existing property set twice in order to define a total of three sets for the class and tool to which the sets apply.
2. All three sets have the same properties as those defined in the original set. In addition, several new properties are added to the second set and several other new properties are added to the third set.

Deleting a Property Set

How To

To delete an entire property set from a model, use the DefaultModelProperties.DeleteDefaultPropertySet method.

Example

```
Sub DeleteDefaultProperties (theModel As Model)
   Dim DefaultProps As DefaultModelProperties
   Set DefaultProps = theModel.DefaultProperties
   myClass$ = theModel.RootCategory.GetPropertyClassName ()

      b = DefaultProps.DeleteDefaultPropertySet (myClass$,
      myTool$, "SecondSet")
      b = DefaultProps.DeleteDefaultPropertySet (myClass$,
      myTool$, "ThirdSet")

      b = theModel.RootCategory.SetCurrentPropertySetName
      (myTool$, "default")
End Sub
```

Notes on the Example

1. The Element.GetPropertyClassName retrieves the valid internal class name to pass as a parameter on the delete calls.
2. Each DefaultModelProperties.DeleteDefaultPropertySet call deletes a property set from the model.
3. The Element.SetCurrentPropertySetName call sets the tool's current property set to its original set, which happens to be called default.

Getting and Setting the Current Property Set

How To

To find out which property set is the current set for a tool, use the Element.GetCurrentPropertySetName method.

To set the current property set to a particular set name, use the Element.SetCurrentPropertySetName and specify the set of your choice.

Note: When setting the current property set, you must supply a set name that is valid for the specified tool. To retrieve a list of valid set names for a tool, use Element.GetDefaultSetNames.

Example

```
Sub RetrieveElementProperties (theElement As Element)
   Dim AllTools As StringCollection
   Dim theProperties As PropertyCollection
   Dim theProperty As Property
   Set AllTools = theElement.GetToolNames ()
   For ToolID = 1 To AllTools.Count
      ThisTool$ = AllTools.GetAt (ToolID)
      theSet$ = theElement.GetCurrentPropertySetName (ThisTool$)
      Set theProperties = theElement.GetToolProperties (ThisTool$)
      For PropID = 1 To theProperties.Count
      Set theProperty = theProperties.GetAt (PropID)
      Next PropID
   Next ToolID
End Sub
```

Notes on the Example

1. GetToolNames retrieves the tool names that apply to the model element type called Element and returns them as a string collection called AllTools.
2. The current property set is retrieved for each tool name.
3. GetToolProperties retrieves the property collection that belongs to the current tool.
4. Each property that belongs to the tool's property collection is retrieved.

Creating a User-Defined Property Type

How To

Rose Extensibility predefines the following set of property types:

- String
- Integer
- Float
- Char
- Boolean
- Enumeration

When you add properties to a set, you specify one of these types.

In addition, you can define your own property types and add properties of that type to a property set.

To create a user-defined property type, add a property whose type is enumeration and whose value is a string that defines the possible values for the enumeration.

Once you have defined the new type, adding a property of this new type is like adding any other type of property.

Example

```
Sub AddDefaultProperties (theModel As Model)
   Dim DefaultProps As DefaultModelProperties
   Set DefaultProps = theModel.DefaultProperties
   myClass$ = theModel.RootCategory.GetPropertyClassName ()
   b = DefaultProps.AddDefaultProperty (myClass$, "myTool",
      "Set1", "MyNewEnumeration", "Enumeration",
      "Value1,Value2,Value3")
   b = DefaultProps.AddDefaultProperty (myClass$, "myTool",
```

```
      "Set1", "MyEnumeratedProperty", "MyNewEnumeration",
      "Value1")
   b = DefaultProps.AddDefaultProperty (myClass$, "myTool",
      "Set1", "isAppropriate", "Boolean", "True")
   b = DefaultProps.AddDefaultProperty (myClass$, "myTool",
      "Set1", "mySpace", "Integer", "5")
End Sub

Sub Main
   AddDefaultProperties (RoseApp.CurrentModel)
End Sub
```

Notes on the Example

1. This example uses Element.GetPropertyClassName to retrieve the internal name of the class to which the property type will apply.
2. The first AddDefaultProperty call adds the enumeration and defines its possible values in the string “Value1, Value2, Value3”.
3. The second AddDefaultProperty call adds a new property of the new enumerated type; the property value is set to “Value1”.
4. If you want a new type to appear in the specification dialog box in the Rose user interface, you must actually add a property of that type to the set. Using the above example, if you simply created the type MyNewEnumeration, but did not add the property MyEnumeratedProperty, MyNewEnumeration would not appear in the **Type** drop-down list. Once you add the actual property, MyNewEnumeration would appear in the list of types.

Creating a New Tool

There is no explicit way to add a new tool (tab) to a model. However, when you create a new property set or add a new property to a model, you must specify the tool to which the property or set applies. If the tool you specify does not already exist, it will be added during the create or add process.

Placing Classes in Categories

- To create a new class and place it in a category, use the Category.AddClass method.

- To relocate an existing class from one category to another, use the Category.RelocateClass method.

Using Type Libraries for Rose Automation

How To

When you specify an REI class in an automation environment, you must add the prefix **Rose** to the class name, unless the word Rose is already part of the REI class name.

For more information on using Type Libraries with Rose, see *Rose Extensibility Type Libraries* on page 5.

Example

- In Rose Script, the syntax for retrieving the Root Category of a model (that is, its logical view) is:

```
Model.RootCategory
```

- In Rose Automation, the syntax for retrieving the Root Category of a model is:

```
RoseModel.RootCategory
```

- In both Rose Script and Rose Automation, the syntax for retrieving the documentation belonging to a RoseItem is:

```
RoseItem.Documentation
```

Working with Controllable Units

Working with controllable units allows you to divide a model into smaller units. This is particularly useful for multi-user development, as well as for placing a model under configuration management.

The methods that apply to working with controllable units are:

- ControllableUnit.Control method, which associates a controllable unit with a file name, so that it can be passed to a configuration management application.
- ControllableUnit.Uncontrol method, which removes the file association from the unit.
- ControllableUnit.Load and Unload methods, which load or unload parts of a model (for example, the units for which a person is responsible).

- ControllableUnit.Save or ControllableUnit.SaveAs methods, which actually write the specified controllable unit to a file.

Note: When you save a model, that will also save its controllable units.

Working with Rose Diagrams

Each kind of Rose diagram (class, component, scenario, and so on) inherits from the Diagram class.

A diagram is made up of RoseItem and RoseItemView objects. A RoseItemview object is the visual representation of the actual RoseItem object. As such, it is an object with properties and methods that define its appearance in the diagram window (position, color, size, and so on). You can define multiple RoseItemView objects for any given RoseItem object.

- Use the Diagram.ItemViews method to iterate through the collection of RoseItemView objects belonging to a diagram.
- Use the Diagram.Items method to iterate through the RoseItem objects that exist in the diagram.
- Use the Diagram.GetViewFrom method to find the first RoseItemView object of a given RoseItem object.

  Note: You can only use the GetViewFrom method to retrieve the first RoseItemView object defined for the RoseItem object. Even if you have more than one RoseItemView object, you always only get the first.
- To find out which RoseItemView objects are currently selected in a diagram, iterate through the diagram's RoseItemView collection. As you retrieve each RoseItemView object, use the ItemView.IsSelected method to find out whether the RoseItemView object is currently selected in the diagram. You can then retrieve the selected RoseItemView object, or do any other processing you wish to do based on whether the RoseItemView object is selected.
- A short way to retrieve all selected RoseItemView objects from a diagram is to use the Diagram.GetSelectedItems method. Instead of iterating through the diagram and checking each RoseItemView object, this method simply returns everything that is selected.

Getting an Element from a Collection

There are three ways to get an individual model element from a collection:

- Use the GetwithUniqueID method to directly access the element.
- Iterate through the collection using the element's name using FindFirst, FindNext, and GetAt.
- Iterate through the collection using Count followed by GetAt.

For more information, check the online Help for *Collection Properties and Methods*.

Accessing Collection Elements by Count

How To

To access collection elements by count:

1. Iterate through the collection using the Count property.
2. Retrieve the specific element using the GetAt method when the specific element is found.

Example

```
Dim AllClasses As ClassCollection
Dim theClass As Class
For ClsID = 1 To AllClasses.Count
Set theClass = AllClasses.GetAt (ClsID)
' ToDo: Add your code here...
Next ClsID
```

Accessing Collection Elements by Unique ID

How To

The most direct and easiest way to get an element from within a collection is by unique ID. To access collection elements by unique ID:

1. Use the GetUniqueID method to obtain the element's unique ID.
2. Use the GetwithUniqueID method, specifying the ID you obtained in step 1.

Example

```
Dim theClasses As ElementCollection
Dim theClass As Element
theID =theClasses.theClass.GetUniqueID ()
theClass = theClass.GetwithUniqueID (theID)
```

Accessing Collection Elements by Name

How To

To access an operation belonging to a class:

1. Use FindFirst to find the first occurrence of the specified operation in the collection.
2. Use FindNext to iterate through subsequent occurrences of the operation.
3. Retrieve the specific operation using the GetAt method when the specific operation is found.

Example

```
Sub PrintOperations (theClass As Class, OperationName As
    String)
   Dim theOperation As Operation
   OperID = theClass.Operations.FindFirst (OperationName$)
   Do
      Set theOperation = theClass.Operations.GetAt (OperID)
      ' ToDo:  Add your code here...
      OperID = theClass.Operations.FindNext (OperID,
        OperationName$)
   Loop Until OperID = 0
End Sub
```

Using the Rational Rose Script Editor

4

Contents

This chapter is organized as follows:

- *The Rose Script Editor* on page 53
- *The Script Editor Window* on page 54
- *Opening a Script* on page 54
- *Creating New Rose Scripts* on page 55
- *Selecting a Font for the Script Editor* on page 55
- *Moving the Insertion Point in a Script* on page 56
- *Selecting Text* on page 57
- *Deleting, Cutting, Copying, and Pasting Text* on page 59
- *Adding Comments to a Script* on page 59
- *Finding and Replacing Text* on page 60
- *Running, Pausing, and Stopping Your Script* on page 62
- *Tracing Script Execution* on page 63
- *Setting and Removing Breakpoints* on page 64
- *Working with Watch Variables* on page 66
- *Compiling Your Script* on page 69
- *Using Interscript Calls* on page 70
- *Working with the Dialog Editor* on page 70

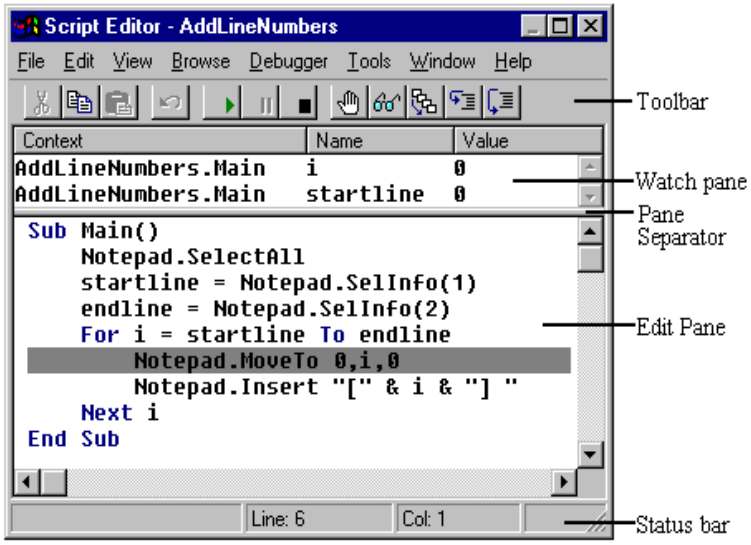
The Rose Script Editor

The Rational Rose Script Editor provides an integrated environment for creating, debugging, and compiling scripts that work with the Rose Extensibility Interface.

The Script Editor Window

Figure 7 on page 54 shows the Script Editor application window.

Figure 7 Rose Script Editor



The Script Editor's application window contains the following elements:

- Toolbar: a collection of tools that you can use to provide instructions to the Script Editor
- Edit pane: a window containing the source code for the script you are currently editing
- Watch pane: a window that opens to display the watch variable list after you have added one or more variables to that list
- Pane separator: a divider that appears between the edit pane and the watch pane when the watch pane is open
- Status bar: displays the current location of the insertion point within your script

Opening a Script

To open a script in the Script Editor:

1 Click **Tools > Open Script**.

2 Select the script to open and select **OK**.

The script is displayed in a new Script Editor window.

Creating New Rose Scripts

Creating a New Script from Scratch

To create a new script in the Script Editor:

1. Click **Tools > New Script**.
2. Enter your script in the new Script Editor window.
3. Enter your script text.
4. Click **File > Save As** and save the new script.

Creating a New Script from an Existing Script

To create a new script from an existing script:

1. Click **Tools > Open Script**.
2. Select a file from the list of available scripts.
3. Click **OK** to enter the Script Editor and display the script.
4. Select the script text and click **Copy** to save the script text to the Clipboard.
5. Click **Tools > New Script**.
6. Click **Paste** to paste the existing script text into the new script window.
7. Click **File > Save As** and save the new script.

Selecting a Font for the Script Editor

When you create a new script or edit an existing script, you can select the text font in the Watch and Edit panes of the Script Editor window.

To select a font for the Script Editor:

1. To make sure the Script Editor window has the focus, do one of the following:
   - Create a new script (**Tools > New Script**).
   - Edit an existing script (**Tools > Open Script**).
   - Click on an already open Script Editor window.
2. Click **Edit > Font** to display the **Font** dialog box.

3 Select the font, font style, size, and script. (For information about each option, click the question mark, and then click the option.)

   A sample of the selected font appears in the **Sample** box.

4 Click **OK**.

   The text in the Watch and Edit panes of the Script Editor appears in the selected font.

Moving the Insertion Point in a Script

There are two ways to move the insertion point in a script:

- With the mouse
- By specifying a line number

Moving the Insertion Point with the Mouse

This approach is useful when the area of the screen to which you want to move the insertion point is currently visible.

To move the insertion point with the mouse:

1 Use the scroll bars at the right and bottom of the display to scroll the target area of the script into view if it is not already visible.

2 Place the mouse pointer where you want to position the insertion point.

3 Click the left mouse button.

   The insertion point is repositioned.

Note: When you scroll the display with the mouse, the insertion point remains in its original position until you reposition it with a mouse click. If you attempt to perform an editing operation when the insertion point is not in view, the Script Editor automatically scrolls the insertion point into view before performing the operation.

Moving the Insertion Point to a Specified Line in Your Script

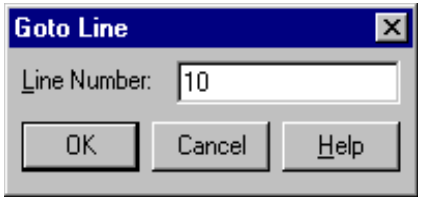
This approach is useful when the area of the screen to which you want to move the insertion point is not currently visible but you know the number of the target line.

To move the insertion point to a specified line:

1 Click **Edit > Goto Line**.

The Script Editor displays the **Goto Line** dialog box.

Figure 8 Goto Line Dialog Box



2 Enter the number of the line in your script to which you want to move the insertion point.

3 Click **OK** or press ENTER.

4 The insertion point is positioned at the start of the line you specified. If that line was not already displayed, the Script Editor scrolls it into view.

Note: The insertion point cannot be moved so far below the end of a script as to scroll the script entirely off the display. When the last line of your script becomes the first line on your screen, the script will stop scrolling, and you will be unable to move the insertion point below the bottom of that screen.

Selecting Text

There are three ways to select text in an open script:

- With the mouse
- With the keyboard
- By selecting an entire line

Selecting Text with the Mouse

To use the mouse to select text in your script:

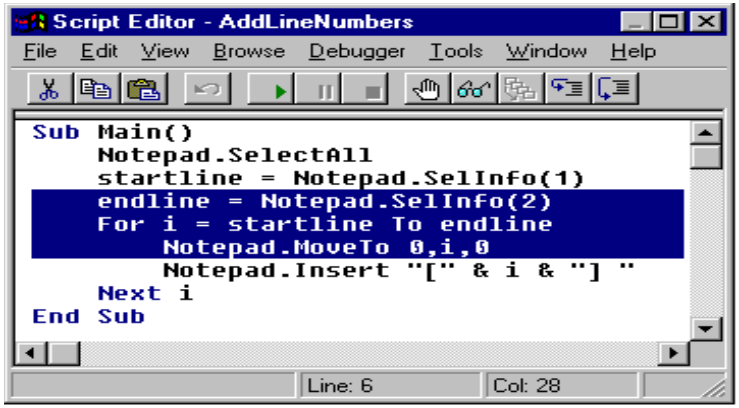
1 Place the mouse pointer where you want your selection to begin.

2 Do one of the following:

- While pressing the left mouse button, drag the mouse until you reach the end of your selection, and release the mouse button.
- While pressing SHIFT, place the mouse pointer where you want your selection to end and click the left mouse button.

The selected text is highlighted on your display.

Figure 9 Selected Script Text



Selecting Text with the Keyboard

To use keyboard shortcuts to select text in your script:

1 Place the insertion point where you want your selection to begin.

2 While pressing SHIFT, use one of the navigating keyboard shortcuts to extend the selection to the desired ending point.

The selected text is highlighted on your display.

Selecting an Entire Line

To use the keyboard to select one or more whole lines in your script:

1 Place the insertion point at the beginning of the line you want to select.

2 Press SHIFT + DOWN ARROW.

The entire line, including the end-of-line character, is selected.

3 To extend your selection to include additional whole lines of text, repeat step 2.

Deleting, Cutting, Copying, and Pasting Text

Deleting Text

Do one of the following to remove characters, selected text, or entire lines from your script:

- To remove a single character to the left of the insertion point, press BACKSPACE once.
- To remove a single character to the right of the insertion point, press DELETE once.
- To remove multiple characters, hold down BACKSPACE or DELETE.
- To remove text that you have selected, press BACKSPACE or DELETE.
- To remove an entire line, place the insertion point in that line and press CTRL+Y.

Cutting a Selection

To cut text from your script and place it on the Clipboard, press CTRL+X.

Copying a Selection

To copy text from your script and place it on the Clipboard, press CTRL+C.

Pasting the Contents of the Clipboard into Your Script

To paste the contents of the Clipboard into your script:

1. Position the insertion point where you want to place the contents of the Clipboard.
2. Press CTRL+V.

Adding Comments to a Script

There are two types of comments you can add to a script:

- Full-Line Comment
- Comment at the End of a Line of Code

Adding a Full-Line Comment

To designate an entire line as a comment:

1. Type an apostrophe (') at the start of the line.
2. Type your comment following the apostrophe.

When your script is run, the presence of the apostrophe at the start of the line will cause the entire line to be ignored.

Adding a Comment at the End of a Line of Code

To designate the last part of a line as a comment:

1. Position the insertion point in the empty space beyond the end of the line of code.
2. Type an apostrophe (').
3. Type your comment following the apostrophe.

When your script is run, the code on the first portion of the line will be executed, but the presence of the apostrophe at the start of the comment will cause the remainder of the line to be ignored.

Finding and Replacing Text

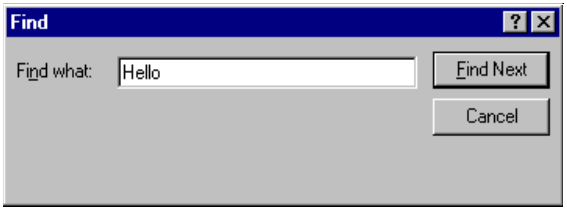
Finding Specified Text

To locate instances of specified text quickly anywhere within your script:

1. Move the insertion point to where you want to start your search. To start at the beginning of your script, press CTRL+HOME.
2. Press CTRL+F.

   The Script Editor displays the **Find** dialog box.

Figure 10 Find Dialog Box



3. In the **Find what** box, specify the text you want to find or select it from the list of previous searches.
4. Click **Find Next** or press ENTER.

   The **Find** dialog box remains displayed, and the Script Editor either highlights the first instance of the specified text or indicates that it cannot be found.

5 If the specified text has been found, repeat step 4 to search for the next instance of it.

Note: If the **Find** dialog box blocks your view of an instance of the specified text, you can move the dialog box out of your way and continue with your search.

6 To remove the **Find** dialog box while maintaining the established search criteria, click **Cancel**.

7 Press F3 to find successive occurrences of the specified text.

Note: If you press F3 when you have not previously specified text for which you want to search, the Script Editor displays the **Find** dialog box so you can specify the desired text.

Replacing Specified Text

To automatically replace either all instances or selected instances of specified text:

1 Move the insertion point to where you want to start the replacement operation. To start at the beginning of your script, press CTRL+HOME.

2 Click **Edit > Replace**.

The Script Editor displays the **Replace** dialog box.

Figure 11 Replace Dialog Box

3 In the **Find what** box, specify the text you want to replace or select it from the list of previous searches.

4 In the **Replace with** box, specify the replacement text or select it from the list of previous replacements.

5 To replace selected instances of the specified text, click **Find Next**.

The Script Editor either highlights the first instance of the specified text or indicates that it cannot be found.

6 If the specified text has been found, either click **Replace** to replace that instance of it or click **Find Next** to highlight the next instance (if any).

Each time you click **Replace**, the Script Editor replaces that instance of the specified text and automatically highlights the next instance.

Running, Pausing, and Stopping Your Script

Running Your Script

To compile and run your script from within the Script Editor, click **Go** on the toolbar or press F5.

The script is compiled (if it has not already been compiled), the focus is switched to the parent window, and the script is executed.

Note: During script execution, the Script Editor's application window is available only in a limited manner. Some of the menu commands may be unavailable, and the toolbar tools may be inoperative.

You can also use the Application Class ExecuteScript method to run scripts. See the online Help for details.

Pausing an Executing Script

To suspend the execution of a script that you are running, press CTRL+BREAK.

Execution of the script is suspended, and the instruction pointer (a gray highlight) appears on the line of code where the script stopped executing.

Note: The instruction pointer designates the line of code that will be executed next if you resume running your script.

Stopping an Executing Script

To stop the execution of a script that you are running:

1 If it is not paused, pause the script.

2 Click **StopDebugging** on the toolbar or press SHIFT+F5.

Note: Many of the functions of the Script Editor's application window may be unavailable while you are running a script. If you want to stop your script, but find that the toolbar is currently inoperative, press CTRL+BREAK to pause your script, then click **StopDebugging**.

Tracing Script Execution

Stepping Through Your Script

To trace the execution of your script with either the **StepInto** or **StepOver** method:

1 Do one of the following:

- Click **StepInto** or **StepOver** on the toolbar.
- Press F11(**StepInto**) or F10 (**StepOver**).

The Script Editor places the instruction pointer on the sub main line of your script.

Note: When you initiate execution of your script using either of these methods, the script will first be compiled, if necessary. Therefore, there may be a slight pause before execution actually begins. If your script contains any compile errors, it will not be executed. To debug your script, first correct any compile errors, and then execute it again.

2 To continue tracing the execution of your script, repeat step 1.

Each time you repeat step 1, the Script Editor executes the line or the procedure that contains the instruction pointer and then moves the instruction pointer to the next line or procedure to be executed.

3 When you finish tracing the execution of your script, either click **Go** on the toolbar (or press F5) to run the script at full speed or click **Stop Debugging** to halt execution of the script.

Displaying the Calls Dialog Box

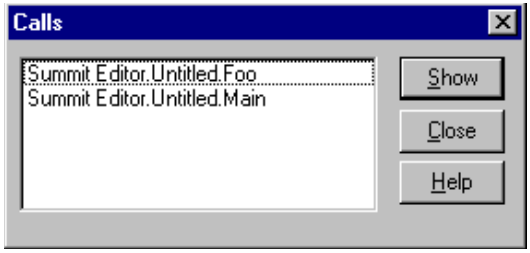
When you are stepping through a subroutine, you may need to determine the procedure calls by which you arrived at that point in your script.

To use the **Calls** dialog box to obtain this information:

1 Click **Calls** on the toolbar.

The Script Editor displays the **Calls** dialog box, which lists the procedure calls made by your script in the course of arriving at the present subroutine.

Figure 12 Script Calls Dialog Box



2 Select the name of the procedure you wish to view.

3 Click **Show**.

The Script Editor highlights the currently executing line in the procedure you selected, scrolling that line into view if necessary. (During this process, the instruction pointer remains in its original location in the subroutine.)

Setting and Removing Breakpoints

You set and remove breakpoints in your script as part of the debugging process.

Starting Debugging Partway Through a Script

To begin the debugging process at a selected point in your script:

1 Place the insertion point in the line where you want to start debugging.

2 To set a breakpoint on that line, click **Toggle Breakpoint** on the toolbar or press F9.

The line on which you set the breakpoint now appears in contrasting type.

3 Click **Go** on the toolbar or press F5.

The Script Editor runs your script at full speed from the beginning and then pauses prior to executing the line containing the breakpoint. It places the instruction pointer on that line to designate it as the line that will be executed next when you either proceed with debugging or resume running the script.

Continuing Debugging at a Line Outside the Current Subroutine

You can continue debugging at a line that is not within the same subroutine.

To move the instruction pointer to that line:

1. Place the insertion point in the line where you want to continue debugging.
2. To set a breakpoint on that line, click **Toggle Breakpoint** on the toolbar or press F9.

   The line on which you set the breakpoint now appears in contrasting type.
3. Click **Go** on the toolbar or press F5.

   The Script Editor runs your script at full speed from the beginning and then pauses prior to executing the line containing the breakpoint. It places the instruction pointer on that line to designate it as the line that will be executed next. You can now resume stepping through your script from that point.

Debugging Selected Portions of Your Script

You can use breakpoints if you only need to debug parts of your script.

To debug selected portions of your script by using breakpoints:

1. Place a breakpoint at the start of each portion of your script that you want to debug.

   Note: Up to 255 lines in your script can contain breakpoints.
2. Click **Go** on the toolbar or press F5.

   The script executes at full speed until it reaches the line containing the first breakpoint and then pauses with the instruction pointer on that line.
3. Step through as much of the code as you need to.
4. To resume running your script, click **Go** on the toolbar or press F5.

   The script executes at full speed until it reaches the line containing the second breakpoint and then pauses with the instruction pointer on that line.
5. Repeat steps 3 and 4 until you have finished debugging the selected portions of your script.

Removing a Single Breakpoint Manually

To delete breakpoints manually one at a time:

1 Place the insertion point on the line containing the breakpoint that you want to remove.

2 Click **Toggle Breakpoint** on the toolbar or press F9.

The breakpoint is removed, and the line no longer appears in contrasting type.

Removing All Breakpoints Manually

To delete all breakpoints manually in a single operation, click **Debugger > Clear All Breakpoints**.

Working with Watch Variables

Watch variables allow you to track the changing values of variables in a script.

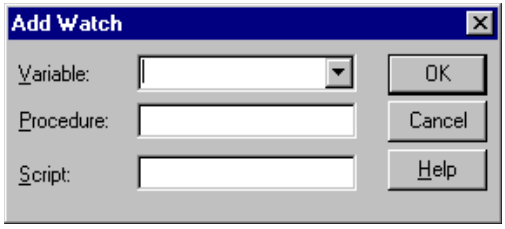
Adding Watch Variables

To add a variable to the Script Editor's watch variable list:

1 Click **Add Watch** on the toolbar or press SHIFT+F9.

The Script Editor displays the **Add Watch** dialog box.

Figure 13 Add Watch Dialog Box



2 In the **Variable** list, enter the name of the variable you want to add to the watch variable list.

You can only watch variables of fundamental data types, such as Integer, Long, Variant, and so on; you cannot watch complex variables such as structures or arrays. You can, however, watch individual elements of arrays or structure members.

Use the following syntax to watch individual elements of arrays or structure members in a script:

`[variable [(index,...)] [.member [(index,...)]]...]`

Where *variable* is the name of the structure or array variable, *index* is a literal number, and *member* is the name of a structure member.

For example, Table 6 on page 67 shows valid watch expressions.

Table 6 Sample Watch Expressions

Watch Variable	Description
a(1)	Element 1 of array a
person.age	Member age of structure person
company(10,23).person.age	Member age of structure person that is at element 10,23 within the array of structures called company

Note: If you are executing the script, you can display the names of all the variables that are "in scope," or defined within the current function or subroutine, in the **Variable** drop-down list and select the variable you want from that list.

3 In the **Procedure** box, enter the name of the RoseScript subroutine or function whose variable you want to add to the watch variable list. For example, **Main** for the Main subroutine, **Area** for the Area function,

4 In the **Script** box, enter the name of the RoseScript without the .ebs extension whose variable you want to add to the watch variable list. For example, **CountClasses** for the CountClasses.ebs RoseScript.

5 Click **OK** or press ENTER.

If this is the first variable you are placing on the watch variable list, the watch pane opens far enough to display that variable. If the watch pane was already open, it expands far enough to display the variable you just added.

Note: Although you can add as many watch variables to the list as you want, the watch pane only expands until it fills half of the Script Editor's application window. If your list of watch variables becomes longer than that, you can use the watch pane's scroll bars to bring hidden portions of the list into view.

Selecting Variables on the Watch List

To delete a variable from the Script Editor's watch variable list or modify the value of a variable on the list, do one of the following:

- Place the mouse pointer on the variable you want to select and click the left mouse button.
- If one of the variables on the watch list is already selected, use the arrow keys to move the selection highlight to the desired variable.
- If the insertion point is in the edit pane, press F6 to highlight the most recently selected variable on the watch list and then use the arrow keys to move the selection highlight to the desired variable.

  Note: Pressing F6 again returns the insertion point to its previous position in the edit pane.

Deleting Watch Variables

To delete a selected variable from the Script Editor's watch variable list:

1. Select the variable on the watch list.
2. Click **Debugger > Delete Watch**, or press DELETE.

Modifying the Value of Variables on the Watch Variable List

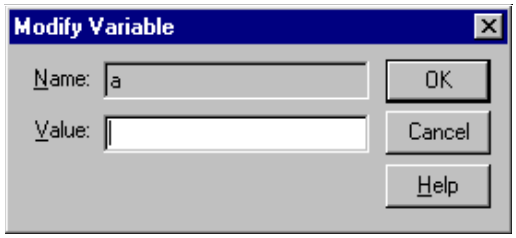
When the debugger has control, you can modify the value of any of the variables on the Script Editor's watch variable list.

To change the value of a selected watch variable.

1. Do one of the following:
   - Place the mouse pointer on the name of the variable whose value you want to modify and double-click the left mouse button.
   - Select the name of the variable whose value you want to modify and press ENTER or F2.

The Script Editor displays the **Modify Variable** dialog box.

Figure 14 Modify Variable Dialog Box



Note: The name of the variable you selected on the watch variable list appears in the **Name** box.

When you use the **Modify Variable** dialog box to change the value of a variable, you do not have to specify the context. The Script Editor first searches locally for the definition of that variable, then privately, then publicly.

2 In the **Value** box, enter the new value for your variable.

3 Click **OK**.

The new value of your variable appears on the watch variable list.

Compiling Your Script

To create compiled script files from your script source:

1 Click **Tools > Open Script** and select the file that contains the script you want to compile.

2 Click **Debugger > Compile** or press F7.

3 Enter the name of the file in which to save the compiled script and select **OK**.

The script is compiled and saved in a file with a .ebx extension.

Note: You can also use the Application.CompileScriptFile method to compile scripts. See the online Help for more details.

Using Interscript Calls

Guidelines for Using a Script to Call Another Script

You can write a script that includes code that calls and executes another script. The following guidelines apply to this process:

- You can only call and execute a compiled script from within another script.
- Use the LoadScript method to load the script into memory.
- Use the FreeScript method to unload the script from memory.
- Even if you call LoadScript multiple times, the script is only loaded into memory one time. However, for each LoadScript call you make, you must include a corresponding FreeScript call. If you do not do this, the script will not be unloaded from memory.

Debugging Interscript Calls

To debug a script that uses interscript calls:

1. Enter the call to the compiled script you are including and set a breakpoint on the call.
2. Click **Debugger > StepInto**.

   The Script Editor displays the source code for the compiled script you are calling, and steps through it line by line.

   When the trace of the called script is complete, the Script Editor redisplays the calling script.

Note: The script you are calling must be compiled with debugging turned on. See *Compiling Your Script* on page 69 for details.

Working with the Dialog Editor

Inserting a Dialog Box into Your Script

To insert a dialog box into your script:

1. Place the insertion point where you want the BasicScript code for the dialog box to appear in your script.
2. Click **Edit > Insert Dialog**.

The Script Editor's application window is temporarily unavailable, and Dialog Editor appears, displaying a new dialog box in its application window.

3 Use the Dialog Editor to create your dialog box.

4 Click **File > Exit and Return** from the Dialog Editor menu to return to the Script Editor.

The Script Editor automatically places the code for the dialog box in your script at the location of the insertion point.

Editing an Existing Dialog Box

To edit an existing dialog box template in your script:

1 Select the BasicScript code for the entire dialog box template.

2 Click **Edit > Edit Dialog**.

The Script Editor's application window is temporarily unavailable, and the Dialog Editor appears, displaying in its application window a dialog box created from the code you selected.

a Use the Dialog Editor to modify your dialog box.

b Click **File > Exit and Return** from the Dialog Editor menu to return to the Script Editor.

The Script Editor automatically replaces the BasicScript code you originally selected with the revised code generated by the Dialog Editor.

Displaying and Adjusting the Grid

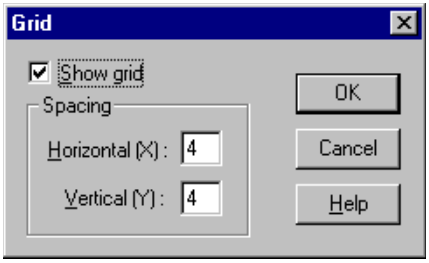
The X and Y settings help you position controls more precisely within your dialog box. The values of X and Y in the **Grid** dialog box determine the grid's spacing. Assigning smaller X and Y values produces a more closely spaced grid, which enables you to move the mouse pointer in smaller horizontal and vertical increments as you position controls. Assigning larger X and Y values produces the opposite effect on both the grid's spacing and the movement of the mouse pointer. The X and Y settings entered in the **Grid** dialog box remain in effect regardless of whether you choose to display the grid.

To display and adjust the grid:

1 Press CTRL+G.

The Dialog Editor displays the **Grid** dialog box.

Figure 15 Grid Dialog Box



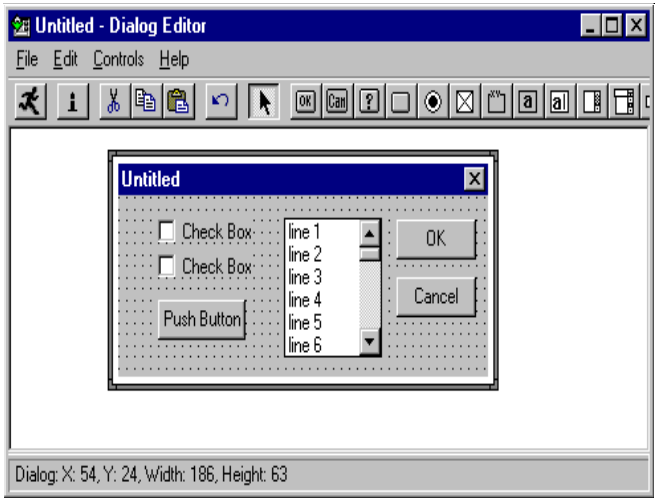
2 To display the grid in your dialog box, select the **Show grid** check box.

3 Enter new values in the **Horizontal (X)** and **Vertical (Y)** boxes.

4 Click **OK** or press ENTER.

The Dialog Editor displays the grid with the settings you specified.

Figure 16 Dialog Editor with Grid Displayed



5 With the grid displayed, line up the crosshairs on the mouse pointer with the dots on the grid to position controls precisely and align them with other controls.

Changing Titles and Labels

To change the title of a dialog box, as well as the labels of group boxes, option buttons, push buttons, text controls, and check boxes:

1. Display the **Information** dialog box for the dialog box whose title you want to change or for the control whose label you want to change.
2. Enter the new title or label in the **Text$** box.

   Note: Dialog box titles and control labels are optional. Therefore, you can leave the **Text$** box blank.
3. If the information in the **Text$** box should be interpreted as a variable name rather than a literal string, select the **Variable Name** check box.
4. Click **OK** or press ENTER.

   The new title or label is now displayed on the title bar or on the control.

Assigning Accelerator Keys

To designate a letter from a control's label to serve as the accelerator key for that control.

1. Display the **Information** dialog box for the control to which you want to assign an accelerator key.
2. In the **Text$** box, type an ampersand (&) before the letter you want to designate as the accelerator key.
3. Click **OK** or press ENTER.

   The letter you designated is now underlined on the control's label, and users will be able to access the control by pressing ALT + the underlined letter.

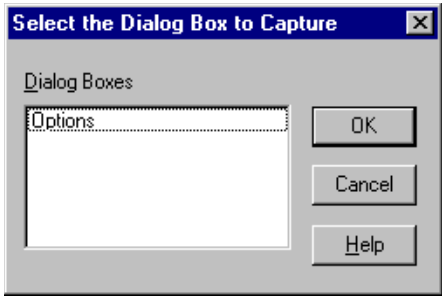
Capturing Standard Windows Dialog Boxes

To capture the standard Windows controls from any standard Windows dialog box in another application, and insert those controls into the Dialog Editor for editing:

1. Display the dialog box you want to capture.
2. Open the Dialog Editor.
3. Click **File > Capture Dialog**.

The Dialog Editor displays the **Select the Dialog Box to Capture** dialog box.

Figure 17 Capturing a Dialog Box



4 Select the dialog box that you want to capture, then click **OK**.

Note: The Dialog Editor only supports standard Windows controls and standard Windows dialog boxes. Therefore, if the target dialog box contains both standard Windows controls and custom controls, only the standard Windows controls will appear in the Dialog Editor's application window. If the target dialog box is not a standard Windows dialog box, you will be unable to capture the dialog box or any of its controls.

Testing Your Dialog Boxes

The Dialog Editor lets you run your edited dialog box for testing purposes. When you click **Test Dialog**, your dialog box becomes functional, which gives you an opportunity to make sure it functions properly and fix any problems before you incorporate the dialog box template into your script.

Before you run your dialog box, take a moment to look it over for basic problems such as the following:

- Does the dialog box contain a command button—that is, a default **OK** or **Cancel** button, a push button, or a picture button?
- Does the dialog box contain all the necessary push buttons?
- Does the dialog box contain a **Help** button if one is needed?
- Are the controls aligned and sized properly?
- If there is a text control, is its font set properly?
- Are the close box and title bar displayed (or hidden) as you intended?
- Are the control labels and dialog box title spelled and capitalized correctly?
- Do all the controls fit within the borders of the dialog box?

- Could you improve the design of the dialog box by adding one or more group boxes to set off groups of related controls?
- Could you clarify the purpose of any unlabeled control (such as a text box, list box, combo box, drop-down list, picture, or picture button) by adding a text control to serve as a label for it?
- Have you made all the necessary accelerator key assignments?
- After you have fixed any elementary problems, you are ready to run your dialog box so you can check for problems that do not become apparent until a dialog box is activated.

Testing your dialog box is an iterative process that involves running the dialog box to see how well it works, identifying problems, stopping the test, and fixing those problems. You can then run the dialog box again to make sure the problems are fixed and to identify any additional problems, and do so until the dialog box functions the way you intend.

To test your dialog box and fine-tune its performance:

1. Click **Test Dialog** on the toolbar, or press F5, to make the dialog box operational.
2. Check the dialog box's functions.
3. To stop the test, click **Test Dialog**, press F5, or click the dialog box's close box (if it has one).
4. Make any necessary adjustments to the dialog box.
5. Repeat steps 1-4 as many times as you need in order to get the dialog box working properly.

Incorporating Dialog Boxes or Controls into Your Script

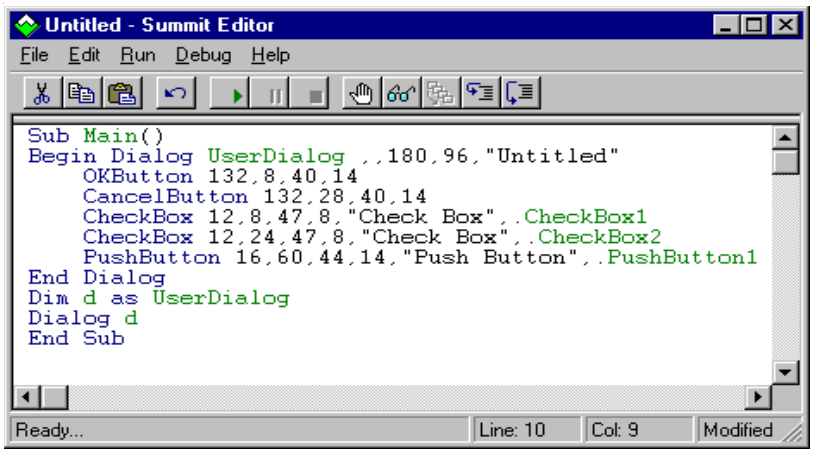
You create dialog boxes and dialog box controls in the Dialog Editor. To incorporate them into a script, you copy them to the Clipboard. When you copy the dialog box to the Clipboard, it is stored in the form of Basic Script statements. You then paste the contents of the Clipboard into the script.

To incorporate a dialog box or control into your script:

1. Select the dialog box or control that you want to incorporate into your script.
2. Press CTRL+C.
3. Open your script and paste the contents of the Clipboard at the desired point.

The dialog box template or control appears in BasicScript statements in your script, as shown in Figure 18 on page 76.

Figure 18 Sample Dialog Box in Basic Script



Selecting Controls

To select a control in a dialog box, do one of the following:

- With the **Pick** tool active, place the mouse pointer on the desired control and click the mouse button.
- With the **Pick** tool active, press the TAB key repeatedly until the focus moves to the desired control.

  The control is now surrounded by a thick frame to indicate that it is selected and you can edit it.

Selecting Dialog Boxes

To select an entire dialog box, do one of the following:

- With the **Pick** tool active, place the mouse pointer on the title bar of the dialog box or on an empty area within the borders of the dialog box (that is, on an area where there are no controls) and click the mouse button.
- With the **Pick** tool active, press the TAB key repeatedly until the focus moves to the dialog box.

  The dialog box is now surrounded by a thick frame to indicate that it is selected and can be edited.

Repositioning Items

Repositioning Items with the Mouse

The increments by which you can move a control with the mouse are governed by the grid setting. For example, if the grid's X setting is 4 and its Y setting is 6, you will be able to move the control horizontally only in increments of 4 X units and vertically only in increments of 6 Y units. This feature is useful if you are trying to align controls in your dialog box. See *Displaying and Adjusting the Grid* on page 71.

To reposition an item in a dialog box or control by dragging it with the mouse:

1. With the **Pick** tool active, place the mouse pointer on an empty area of the dialog box or on a control.
2. Press the mouse button and drag the dialog box or control to the desired location.

Repositioning Items with the Arrow Keys

To reposition an item in a dialog box or control by dragging it with the arrow keys:

1. Select the dialog box or control that you want to move.
2. Do one of the following:
   - Press an arrow key once to move the item by one X or Y unit in the desired direction.
   - Repeatedly press an arrow key to “nudge” the item gradually along in the desired direction.

   Note: When you reposition an item with the arrow keys, a faint, partial afterimage of the item may remain visible in the item's original position. These afterimages disappear once you test your dialog box.

Repositioning Dialog Boxes with the Dialog Information Dialog Box

To reposition items in a dialog box or control by using the **Dialog Information** dialog box:

1. Display the **Dialog Box Information** dialog box.

   Note: For information on displaying the **Dialog Information** dialog box, see *Displaying the Information Dialog Boxes for Dialog Boxes* on page 82.
2. Do one of the following:
   - Change the X and Y coordinates in the **Position** group box.

- Leave the X and / or Y coordinates blank.

3 Click **OK** or press ENTER.

If you specified X and Y coordinates, the dialog box moves to that position. If you left the X coordinate blank, the dialog box will be centered horizontally relative to the parent window of the dialog box when the dialog box is run. If you left the Y coordinate blank, the dialog box will be centered vertically relative to the parent window of the dialog box when the dialog box is run.

Repositioning Controls with the Dialog Information Dialog Box

To move a selected control by changing its coordinates in the **Dialog Information** dialog box for that control:

Note: For information on displaying the **Dialog Information** dialog box, see *Displaying the Information Dialog Boxes for Controls* on page 83.

1 Display the **Information** dialog box for the control that you want to move.

2 Change the X and Y coordinates in the **Position** group box.

3 Click **OK** or press ENTER.

The control moves to the specified position.

Resizing Items

Resizing Items with the Mouse

To change the size of a selected dialog box or control by dragging its borders or corners with the mouse:

1 With the **Pick** tool active, select the dialog box or control that you want to resize.

2 Place the mouse pointer over a border or corner of the item.

3 Depress the mouse button and drag the border or corner until the item reaches the desired size.

Resizing Items with the Information Dialog Box

To change the size of a selected dialog box or control by changing its **Width** or **Height** settings in the **Information** dialog box:

1 Display the **Information** dialog box for the dialog box or control that you want to resize.

2 Change the **Width** and **Height** settings in the **Size** group box.

3 Click **OK** or press ENTER.

The dialog box or control is resized to the dimensions you specified.

Resizing Selected Items Automatically

To adjust the borders of certain controls automatically to fit the text displayed on them:

1 With the **Pick** tool active, select the option button, text control, push button, check box, or text box that you want to resize.

2 Press F2.

The borders of the control will expand or contract to fit the text displayed on it.

Adding Controls

To add one or more controls to your dialog box using simple mouse and keyboard methods.

1 From the toolbar, choose the tool corresponding to the type of control you want to add.

Note: When you pass the mouse pointer over an area of the display where a control can be placed, the pointer becomes an image of the selected control with crosshairs (for positioning purposes) to its upper left. The name and position of the selected control appear on the status bar. When you pass the pointer over an area of the display where a control cannot be placed, the pointer changes into a circle with a slash through it (the "prohibited" symbol).

Note: You can only insert a control within the borders of the dialog box you are creating. You cannot insert a control on the dialog box's title bar or outside its borders.

2 Place the pointer where you want the control to be positioned and click the mouse button.

The control you just created appears at the specified location. The upper left corner of the control corresponds to the position of the pointer's crosshairs at the moment you clicked the mouse button. The control is surrounded by a thick frame, which means that it is selected, and it may also have a default label.

After the new control appears, the mouse pointer becomes an arrow, to indicate that the **Pick** tool is active and you can once again select any of the controls in your dialog box.

3 To add another control of the same type as the one you just added, press CTRL+D.

A duplicate copy of the control appears.

4 To add a different type of control, repeat steps 1 and 2.

5 To reactivate the **Pick** tool, do one of the following:

- Click the arrow-shaped tool on the toolbar.
- Click the title bar of the dialog box or outside the borders of the dialog box (that is, on any area where the mouse pointer turns into the "prohibited" symbol).

Duplicating Controls

You can use the Dialog Editor's duplicating feature to create one or more copies of a particular control.

To duplicate controls:

1 Select the control that you want to duplicate.

2 Press CTRL+D.

A duplicate copy of the selected control appears in your dialog box.

3 Repeat step 2 as many times as necessary to create the desired number of duplicate controls.

Adding Pictures to a Dialog Box

You can add pictures to a dialog box from a file or from a picture library.

Adding Pictures from Files

You can display a Windows bitmap or metafile from a file on a picture control or picture button control by using the control's **Information** dialog box to indicate the file in which the picture is contained.

To add pictures from files:

1 Display the **Information** dialog box for the picture control or picture button control whose picture you want to specify.

2 In the **Picture source** option button group, click **File**.

3 In the **Name$** box, enter the name of the file containing the picture you want to display in the picture control or picture button control.

Note: By clicking the **Browse** button, you can display the **Select a Picture File** dialog box and use it to find the file.

4 Click **OK** or press ENTER.

The picture control or picture button control displays the picture you specified.

Adding Pictures from Picture Libraries

You can display a Windows bitmap or metafile from a file on a picture control or picture button control by using the control's **Information** dialog box to indicate the file in which the picture is contained.

To add pictures from picture libraries:

1. Display the **Information** dialog box for the picture control or picture button control whose picture you want to specify.
2. In the **Picture source** option button group, click **File**.
3. In the **Name$** box, enter the name of the file containing the picture you want to display in the picture control or picture button control.

   Note: By clicking the **Browse** button, you can display the **Select a Picture File** dialog box and use it to find the file.
4. Click **OK** or press ENTER.

   The picture control or picture button control displays the picture you specified.

Pasting Items into Dialog Editor

Pasting Existing Dialog Boxes into the Dialog Editor

You can modify a BasicScript dialog box template contained in your script by selecting the template and pasting it into the Dialog Editor for editing.

To paste dialog boxes into the Dialog Editor:

1. Copy the entire BasicScript dialog box template (from the Begin Dialog instruction to the End Dialog instruction) from your script to the Clipboard.
2. Open the Dialog Editor.
3. Press CTRL+V.
4. When the Dialog Editor asks whether you want to replace the existing dialog box, click **Yes**.

   The Dialog Editor creates a new dialog box corresponding to the template contained on the Clipboard.

Pasting Controls from Existing Dialog Boxes into the Dialog Editor

You can modify the BasicScript statements in your script that correspond to one or more dialog box controls by selecting the statements and pasting them into Dialog Editor for editing.

To paste controls into the Dialog Editor:

1. Copy the BasicScript description of the control(s) from your script to the Clipboard.
2. Open the Dialog Editor.
3. Press CTRL+V.

   The Dialog Editor adds to your current dialog box one or more controls corresponding to the description contained on the Clipboard.

Displaying the Information Dialog Boxes

There are two types of **Information** dialog boxes:

- Information Dialog Box for Dialogs
- Information Dialog Box for Controls

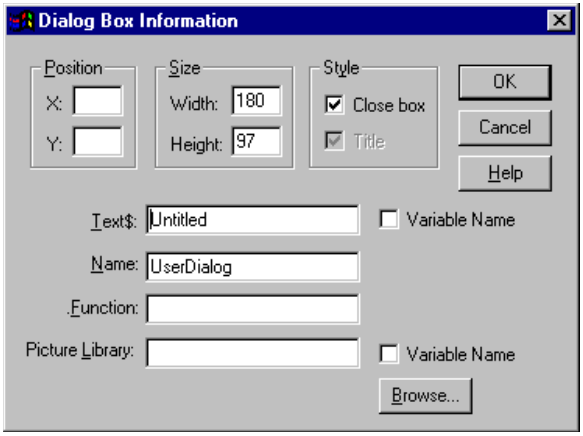
Displaying the Information Dialog Boxes for Dialog Boxes

To display the **Dialog Box Information** dialog box to check and adjust attributes that pertain to the dialog box as a whole, do one of the following:

- With the **Pick** tool active, place the mouse pointer on an area of the dialog box where there are no controls and double-click the mouse button.
- With the **Pick** tool active, select the dialog box and either click the **Information** tool on the toolbar, press ENTER, or press CTRL+I.

The **Dialog Box Information** dialog box appears.

Figure 19 Dialog Box Information Dialog Box



Attributes You Can Adjust with the Dialog Box Information Dialog Box

The **Dialog Box Information** dialog box can be used to check and adjust the following attributes, which pertain to the dialog box as a whole:

- **Position** (optional): X and Y coordinates on the display, in dialog box units
- **Size** (mandatory): width and height of the dialog box, in dialog box units
- **Style** (optional): options that allow you to determine whether the close box and title bar are displayed
- **Text$** (optional): text displayed on the title bar of the dialog box
- **Name** (mandatory): name by which you refer to this dialog box template in your BasicScript code
- **Function** (optional): name of a BasicScript function in your dialog box
- **Picture Library** (optional): picture library from which one or more pictures in the dialog box are obtained

Displaying the Information Dialog Boxes for Controls

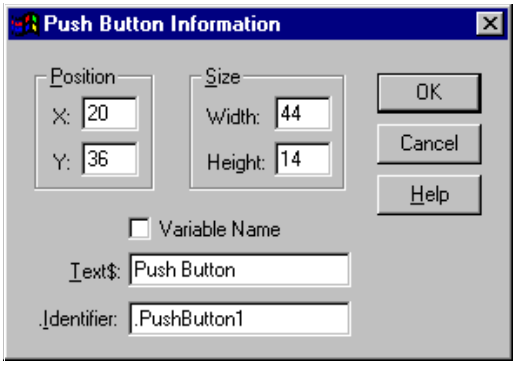
To display the **Information** dialog box for a control to check and adjust attributes that pertain to that particular control, do one of the following:

- With the **Pick** tool active, place the mouse pointer on the desired control and double-click the mouse button.

- With the **Pick** tool active, select the control and either click the **Information** tool on the toolbar, press ENTER, or press CTRL+I.

  The Dialog Editor displays an **Information** dialog box corresponding to the control you selected. For an example, see Figure 20 on page 84.

Figure 20 Control Information Dialog Box



Attributes You Can Adjust with the Information Dialog Boxes for Controls

Control Information dialog boxes can be used to check and adjust the attributes of the following controls:

- **Default OK Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
- **Default Cancel Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
- **Help Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units

- **Size** (mandatory): width and height of the control, in dialog box units
- **FileName$** (optional): Name of the Help file that you want to invoke
- **Context&** (mandatory): The context ID specifying which Help topic to jump to
- **.Identifier** (optional): name by which you refer to a control in your BasicScript code

- **Push Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **Text$** (optional): text displayed on a control
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
- **Option Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **Text$** (optional): text displayed on a control
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
  - **.Option Group** (mandatory): name by which you refer to a group of option buttons in your BasicScript code
- **Check Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **Text$** (optional): text displayed on a control
  - **.Identifier** (mandatory): name by which you refer to a control in your BasicScript code; also contains the result of the control after the dialog box has been processed
- **Group Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units

- **Text$** (optional): text displayed on a control
- **.Identifier** (optional): name by which you refer to a control in your BasicScript code

- **Text Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **Text$** (optional): text displayed on a control
  - **Font** (optional): font in which text is displayed
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
- **Text Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **Multiline** (optional): option that allows you to determine whether users can enter a single line of text or multiple lines
  - **.Identifier** (mandatory): name by which you refer to a control in your BasicScript code; also contains the result of the control after the dialog box has been processed
- **List Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (mandatory): name by which you refer to a control in your BasicScript code; also contains the result of the control after the dialog box has been processed
  - **Array$** (mandatory): name of an array variable in your BasicScript code
- **Combo Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units

- **.Identifier** (mandatory): name by which you refer to a control in your BasicScript code; also contains the result of the control after the dialog box has been processed
- **Array$** (mandatory): name of an array variable in your BasicScript code

- **Drop List Box Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (mandatory): name by which you refer to a control in your BasicScript code; also contains the result of the control after the dialog box has been processed
  - **Array$** (mandatory): name of an array variable in your BasicScript code
- **Picture Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
  - **.Identifier** (optional): name of the file containing a picture that you want to display or the name of a picture that you want to display from a specified picture library
  - **Frame** (optional): option that allows you to display a 3-D frame
- **Picture Button Information** dialog box
  - **Position** (mandatory): X and Y coordinates within the dialog box, in dialog box units
  - **Size** (mandatory): width and height of the control, in dialog box units
  - **.Identifier** (optional): name by which you refer to a control in your BasicScript code
  - **.Identifier** (optional): name of the file containing a picture that you want to display or the name of a picture that you want to display from a specified picture library

Rational Rose Script Editor Shortcuts

A

Contents

This chapter is organized as follows:

- *General Shortcuts* on page 89
- *Navigating Shortcuts* on page 90
- *Editing Shortcuts* on page 90
- *Debugging Shortcuts* on page 91
- *File Menu Shortcuts* on page 92
- *Edit Menu Shortcuts* on page 92
- *Debugger Menu Shortcuts* on page 93

This appendix identifies the shortcuts that can be used with the Rational Script Editor.

General Shortcuts

Table 7 General Shortcuts

Key Name(s)	Description
F1	Provides context-sensitive Help for selected menu commands and variables in the watch pane, for BasicScript terms in the edit pane that have been selected or that contain the insertion point, and for displayed dialog boxes.
CTRL+F	Displays the **Find** dialog box, which allows you to specify text for which you want to search.
F3	Searches for the next occurrence of previously specified text. If you have not previously specified text for which you want to search, displays the **Find** dialog box.
ESC	Deactivates the **Help** pointer if it is active. Otherwise, compiles your script and returns you to the host application.

Navigating Shortcuts

Table 8 Navigating Shortcuts

Key Name(s)	Description
UP ARROW	Moves the insertion point up one line.
DOWN ARROW	Moves the insertion point down one line.
LEFT ARROW	Moves the insertion point left by one character position.
RIGHT ARROW	Moves the insertion point right by one character position.
PAGE UP	Moves the insertion point up by one window.
PAGE DOWN	Moves the insertion point down by one window.
CTRL+PAGE UP	Scrolls the insertion point left by one window.
CTRL+PAGE DOWN	Scrolls the insertion point right by one window.
CTRL+LEFT ARROW	Moves the insertion point to the start of the next word to the left.
CTRL + RIGHT ARROW	Moves the insertion point to the start of the next word to the right.
HOME	Places the insertion point before the first character in the line.
END	Places the insertion point after the last character in the line.
CTRL+HOME	Places the insertion point before the first character in the script.
CTRL+END	Places the insertion point after the last character in the script.

Editing Shortcuts

Table 9 Editing Shortcuts

Key Name(s)	Description
DELETE	Removes the selected text or removes the character following the insertion point without placing it on the Clipboard.
BACKSPACE	Removes the selected text or removes the character preceding the insertion point without placing it on the Clipboard.
CTRL+Y	Deletes the entire line containing the insertion point without placing it on the Clipboard.

Table 9 Editing Shortcuts (continued)

Key Name(s)	Description
TAB	Inserts a tab character.
ENTER	Inserts a new line, breaking the current line.
CTRL+C	Copies the selected text, without removing it from the script, and places it on the Clipboard.
CTRL+X	Removes the selected text from the script and places it on the Clipboard.
CTRL+V	Inserts the contents of the Clipboard at the location of the insertion point.
SHIFT + any navigating shortcut	Selects the text between the initial location of the insertion point and the point to which the keyboard shortcut would normally move the insertion point. (For example, pressing SHIFT + CTRL + LEFT ARROW selects the word to the left of the insertion point; pressing SHIFT+CTRL+HOME selects all the text from the location of the insertion point to the start of your script.)
CTRL+Z	Reverses the effect of the preceding editing change(s).

Debugging Shortcuts

Table 10 Debugging Shortcuts

Key Name(s)	Description
SHIFT+F9	Displays the **Add Watch** dialog box, in which you can specify the name of a BasicScript variable. The Script Editor then displays the value of that variable, if any, in the watch pane of its application window.
ENTER or F2	Displays the **Modify Variable** dialog box for the selected watch variable, which enables you to modify the value of that variable.
F6	If the watch pane is open, switches the insertion point between the watch pane and the edit pane.
CTRL+BREAK	Suspends execution of an executing script and places the instruction pointer on the next line to be executed.
F9	Sets or removes a breakpoint on the line containing the insertion point.
F10	Activates the StepOver command, which executes the next line of a BasicScript script and then suspends execution of the script. If the script calls another BasicScript procedure, BasicScript will run the called procedure in its entirety.

Table 10 Debugging Shortcuts (continued)

Key Name(s)	Description
F11	Activates the StepInto command, which executes the next line of a BasicScript script and then suspends execution of the script. If the script calls another BasicScript procedure, execution will continue into each line of the called procedure.

File Menu Shortcuts

Table 11 File Menu Shortcuts

Key Name(s)	Description
CTRL+W	Compiles your script and returns you to the host application.
CTRL+S	Saves the currently open script.

Edit Menu Shortcuts

Table 12 Edit Menu Shortcuts

Key Name(s)	Description
CTRL+Z	Reverses the effect of the preceding editing change(s).
CTRL+X	Removes the selected text from the script and places it on the Clipboard.
CTRL+C	Copies the selected text, without removing it from the script, and places it on the Clipboard.
CTRL+V	Inserts the contents of the Clipboard at the current position of the insertion point.
CTRL+A	Selects all the text in the edit window.
CTRL+F	Displays the **Find** dialog box, which allows you to specify text for which you want to search. Remembers and allows you to choose from a list of previous search strings.
CTRL+H	Displays the **Replace** dialog box, which allows you to substitute replacement text for instances of specified text. Remembers and allows you to choose from a list of previous search and replace strings.
CTRL+G	Presents the **Goto Line** dialog box, which allows you to move the insertion point to the start of a specified line number in your script.

Debugger Menu Shortcuts

Table 13 Debugger Menu Shortcuts

Key Name(s)	Description
F5	Runs the current script.
CTRL+SHIFT+F5	Restarts the current script beginning with the line at which it was stopped using the Break command.
SHIFT+F5	Stops script execution.
F11	Steps through the script code line by line, tracing into called procedures.
F10	Steps through the script code line by line without tracing into called procedures.
F7	Compiles the current script without executing it.
SHIFT+F9	Displays the **Add Watch** dialog box, in which you can specify the name of a BasicScript variable. That variable, together with its value (if any), is then displayed in the watch pane of the Script Editor's application window.
DELETE	Deletes a selected variable from the watch variable list.
ENTER	Displays the **Modify Variable** dialog box for a selected variable, which enables you to modify the value of that variable.
F9	Toggles a breakpoint on the line containing the insertion point.

Developing Add-Ins for Rational Rose

B

Contents

This chapter is organized as follows:

- *Introduction* on page 95
- *Why Create Add-Ins?* on page 96
- *Types of Add-Ins* on page 97
- *What Is in an Add-In?* on page 97
- *UNIX vs. Windows* on page 99
- *Creating Portable Add-Ins* on page 100
- *How to Develop Add-Ins* on page 101

Introduction

This appendix provides additional information for customers wanting to explore the use of add-ins. However, creation of add-ins is not directly supported by Rational Technical Support. Additional support for add-ins is available through the Rational Unified Solutions Partner Program and Rational University.

For more information on the Rational Unified Solutions Partner Program see http://www.rational.com/corpinfo/partners/.

For training on Rational Rose's REI and add-ins see the *Extending Rose* course from Rational University at http://www.rational.com/university/description/.

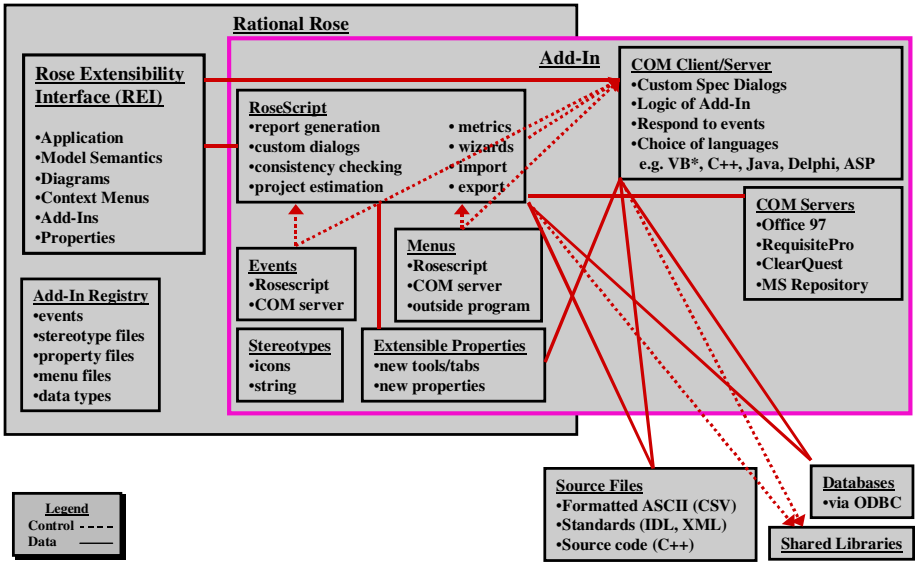
Add-ins allow you to package customizations and automation of several Rose features through the Rose Extensibility Interface (REI) into one package. An add-in is a collection of some combination of the following:

- Main menu items
- Shortcut menu items
- Custom specifications
- Properties
- Data types
- Stereotypes
- Online Help

- Context-sensitive Help
- Event handling
- Functionality through Rose Scripts or controls (OLE-server)

Rose Script or any language such as VB or C++ that can expose itself as an OLE server may be used to build an add-in.

Figure 21 Rose Add-Ins Architecture



Note: Servers that wish to use Rose must use the supplied typelib included with Rose.

Why Create Add-Ins?

You might want to create an add-in as opposed to a script or program if you answer "yes" to any of the following questions:

- Do you want to take advantage of Rose events like OnNewModel and OnAppInit?
- Do you want to interact with other Rose add-ins?

Types of Add-Ins

There are two types of add-ins:

- Basic: A basic add-in is a non-language add-in that supplies its own responses for events to execute third-party scripts or executables, such as a Visual Basic program. It does not use the component view for code generation. A basic add-in cannot register for certain code generation-related events.
- Language: A language add-in takes advantage of the mapping to components by defining a target language. It also supplies its own responses for events that pertain to code generation and round-trip engineering integration. Code generation and round-trip engineering events include OnGenerateCode, OnBrowseBody, and OnBrowseHeader. Language add-ins support custom data types and overriding the default specification.

What Is in an Add-In?

Add-ins customize or contain one or more of the following:

- Main menus
- Shortcut menu
- Custom specifications
- Properties
- Data types
- Stereotypes
- Online Help
- Context-sensitive Help
- Registering for events
- Functionality

Each of these are explained in the next sections.

Main Menus

The Rose main menus are the menus at the top of the Rose window, such as **File** and **Edit**. These menus connect the user interface to functionality in Rose. You can customize these menus to link functionality in your add-in to the Rose user interface.

Shortcut Menu

The Rose shortcut menu appears whenever you or your user right-clicks on part of the user interface. The shortcut menu is another link between the user interface and functionality in Rose. You can customize this menu to link functionality in your add-in to the Rose user interface.

Custom Specifications

Rose displays a standard specification dialog box for each model element to allow definition and description of that model element. If you are writing a language add-in, you can override the standard Rose specification dialog box and display your own custom dialog box. This is useful to:

- Remove irrelevant or inappropriate information.
- Target the dialog box to your end-user's needs.
- Drive the dialog box by stereotype or other characteristics, for example, naming conventions.

Properties

Rose model properties allow you to extend Rose model elements through additional properties and their values. You can add custom tools (such as a tab on the specification dialog box), sets, and properties to store the information relevant to your add-in with each Rose model element. You can also use this information to determine when functionality in your add-in should occur.

Data Types

Rose data types allow you to customize which selections your user sees in the type drop-down list for model elements associated with your add-in.

Stereotypes

Rose stereotypes allow you to customize the look of different model elements as makes sense to your add-in. This custom look can be as simple as an additional text string (for example, <<Special Class>>), or as fancy as new icons for the diagram editor, toolbar buttons, and browser icons.

Online Help

Rose provides extensive online Help to explain the product. You can also add your online Help.

Context-sensitive Help

Rose provides context-sensitive Help to provide quick, brief information on the context. You can add context-sensitive Help to your add-in's user interface, your custom Rose main menu items, shortcut menu items, and properties.

Registering for Events

Rose provides several COM events for which your add-in can register and respond.

Functionality

You can write code to provide the dialog boxes and other functionality desired in your add-in.

UNIX vs. Windows

If you are developing add-ins for UNIX, see Table 14 on page 99 for the differences between UNIX and Windows.

Table 14 UNIX vs. Windows

Windows	UNIX
GUI painter and capture	GUI painter and capture
Custom dialog boxes	Custom dialog boxes
API through COM	MainWin—MainSoft Technology
Rose as COM client	MainWin—MainSoft Technology
ODBC functions	none
GUI drivers (for example, Send Keys)	n/a

The basic difference is that to create an add-in for UNIX, you must "fake" setting up the "registry".

To create a UNIX version:

1. Copy the contents of rose.*version*/addins/cm directory to a new directory in the rose.*version*/addins directory where *version* is the installed version of Rose. For example:

   cd rose.4.5.8153/addins
   mkdir my_addin
   cp cm/* my_addin

2. Change to the new directory. For example:

   cd my_addin

3. Rename cm.mnu and cm.reg to the name of your add-in. For example:

   mv cm.mnu my_addin.mnu
   mv cm.reg my_addin.reg

4. Edit your menu file (.mnu) to add the menus you want. The format is the same on Windows and UNIX.

5. Edit the registry file (.reg) and replace “cm” with the name of your add-in. You should change:

   - HKEY_LOCAL_MACHINE
   - InstallDir
   - MenuFile

   A global search and replace on the document should help.

6. Copy your add-in’s custom Help file to the rose.*version*/help directory where *version* is the installed version of Rose. For example:

   cd rose.4.5.8153/help
   cp /somepath/MyHelpFile.hlp .
   cp /somepath/MyHelpFile.cnt .

Creating Portable Add-Ins

To create add-ins that will be portable to other platforms, keep the following recommendations in mind:

- Keep the logic of the integration in Rose Script.
- Keep dialog boxes and graphical user interfaces (GUIs) in Rose Script.
- Import and export through ASCII files.
- Do not use COM calls—write shell-accessible commands.
- Test the operating system with the BasicScript object (for example, Basic.OS).
- Use path map variables.

How to Develop Add-Ins

The following procedure gives you a high-level look at what you need to do to develop your add-in:

1. Decide which language to use to create your add-in (Rose Script or a COM-enabled language such as Visual Basic, C++).
2. Decide whether you will create a basic or language add-in.
3. Decide which parts of Rose you want to customize or use:
   - Main menus
   - Shortcut menus
   - Custom specifications
   - Properties
   - Data types
   - Stereotypes
   - Online Help
   - Context-sensitive Help
   - Registering for events
4. Design your add-in's functionality.
5. Create all the pieces for your add-in that you need:
   - Menu file (.mnu)
   - Property file (.pty) to add new tools, sets, and properties
   - Data types
   - Stereotypes (.ini, .bmp, .wmf, .emf)
   - Online and context-sensitive Help (.hlp)
   - Method for updating the registry file (.reg)
   - Code to:
     - perform all the functions of your add-in (.ebs, .ebx, .exe, .dll, etc.)
     - register for and handle events
     - create shortcut menu items
     - define your custom specification dialog boxes

- Installation routine
- Uninstallation routine
- Hardcopy documentation
- Anything else specific to your needs

6 Test your add-in and its pieces:

- Installation
- Activation
- New functionality
  - Menu items
  - Shortcut menu items
  - Custom specifications
  - Properties
  - Data types
  - Stereotypes
  - Online Help
  - Context-sensitive Help
  - Events
  - All other add-in functionality
- Deactivation
- Uninstallation

7 Package your add-in and distribute to your customers (whether internal or external).

Working with and customizing each of the items listed above (for example, menus and properties) are explained in more detail in the next sections.

Customizing Main Menus

Each add-in may introduce additions to the menus specific for that add-in, using the menu file technology (*.mnu). This is the only way an add-in can provide main menu items.

For more information on the syntax for the Rose menu file (*.mnu), see *Customizing Rose Main Menus* on page 7.

Note: If you choose to customize the main menus, you must update the registry (discussed in *Updating the Registry* on page 123).

Customizing the Shortcut Menu

For information on customizing the shortcut menu, see *Customizing Rose Shortcut Menus* on page 19.

Creating Custom Specifications

To create and activate custom specifications:

1. Create a language add-in.
2. Register for the OnPropertySpecOpen event.
3. Implement an OnPropertySpecOpen interface in your add-in's OLE server.
4. Code your custom specification dialog boxes.

Customizing Properties

Properties are added to Rose items by add-ins using the existing property file (.pty) technology. Each add-in can optionally supply its own property file that defines a name space for its properties and a tab in the specification editor to hold the custom tool, sets, and properties. You can only define one property file per add-in, but you can define multiple tools, sets, and properties within that one file. The property file is automatically enabled and disabled as your add-in is enabled and disabled. Even when the property file is disabled, however, your custom properties are persisted with the model file. To hide a tab, the user can deactivate the corresponding add-in in Rose.

Design Considerations

The ordering of the tabs (tools) must be independent of when, where, and what add-ins are installed or activated. The tab name (tool name) must be unique for each add-in. Rose cannot detect conflicts. You must always have a "default" set for each of your custom tools.

Note: If you choose to add a property file, you must update the registry (see *Updating the Registry* on page 123).

You can also add, delete, and clone properties through the extensibility interface. For more information on how to do this, see *Managing Default Properties* on page 37 and subsequent sections.

Information in Property Files

Property Files contain the following information:

version

    tool 1

        default set__model element 1

            property 1

            property 2

            ...

            property n

        default set__model element 2

            property 1

            property 2

            ...

            property n

        default set__last model element

            ...

        next set__model element 1

            property 1

            property 2

            ...

            property n

        next set__model element 2

            ...

        next set__last model element

            ...

        last set__model element 1

            ...

        last set__last model element

...

tool 2

default set__model element 1

...

last set__last model element

...

last tool

...

Format for Property Files

This section discusses the format of property files. Keywords are shown in **bold**, while variable information that you need to set is shown in *italics*. Each element is explained at the end of the property file format.

```
# Comments about the property file
# Begin version information
(object Petal
     version    number
     _written   "add-in name"
     charSet    0)
# End version information

# Begin tool definition
(list Attribute_Set
     # Tool setup
     (object Attribute
        tool    "tool"
        name    "propertyID"
        value   "809135966")

     # Begin set and model element definition
     (object    Attribute
        tool    "tool"
        name    "set__model element"
        value   (list Attribute_Set
              # Define first property
              (object Attribute
                 tool    "tool"
                 name    "property"
                 value   datatype)

              # Define second property
              (object Attribute
                 tool    "tool"
                 name    "property"
```

```
                value   datatype)
            ...

            # Define nth property
            (object Attribute
                tool    "tool"
                name    "property"
                value     datatype)
        )
        # End property list
    )
    # End set and model element list

    # Begin next set and model element list
    (
        ...
    )
    # End next set and model element list
)
# End tool definition

# Begin next tool definition. Repeat format.
(
    ...
)
# End next tool definition
# End property file
```

The property file is composed of the following elements:

- Comments: Place a number sign (#) at the beginning of the line to indicate that it is a comment line.
- Number: Enter the petal version number that corresponds to the version of Rose for which you are writing your add-in. To find out what this number is, first locate a model file (.mdl) saved in the same version of Rose. Next, open the model file in a text editor, such as Notepad.
- Add-in name: Enter the name you want to call your add-in. For example, Rose/MyAddin v1.0
- Tool: Enter the name of your tool. For example, My Tool. You may define multiple tools for your add-in in one property file.
- Value: Use the same value (809135966) for each of your tools. If you run into problems, add 1 to the number.
- Set__model element: Enter the name of your set and model element. For example, default__Project, CompilerV1.0__Project, CompilerV2.0__Project, default__Class. You may have multiple sets and multiple model elements per tool. Valid model elements are:

- Association
- Attribute
- Category
- Class
- Has
- Inherit
- Module-Spec
- Module-Body
- Operation
- Param
- Project
- Role
- Subsystem
- Uses

- Property: Enter the name of your property. For example, minCount.
- Data type: Enter the default value for the data type of your property. For example, if your property is:
  - an integer, your default value may be 0.
  - a string, your default value may be “” or “Unknown”.
  - a boolean, your default value may be `TRUE`.

  Table 15 on page 107 lists examples for each of the different data types and how to format them in your property file. Note the cases where quotes are used versus where they are not used.

Table 15 Property File Data Types

Data type	Example and Default	Format

String	Name
	blank
		`(object Attribute`
		`tool "MyTool"`
		`name "Name"`
		`value "")`

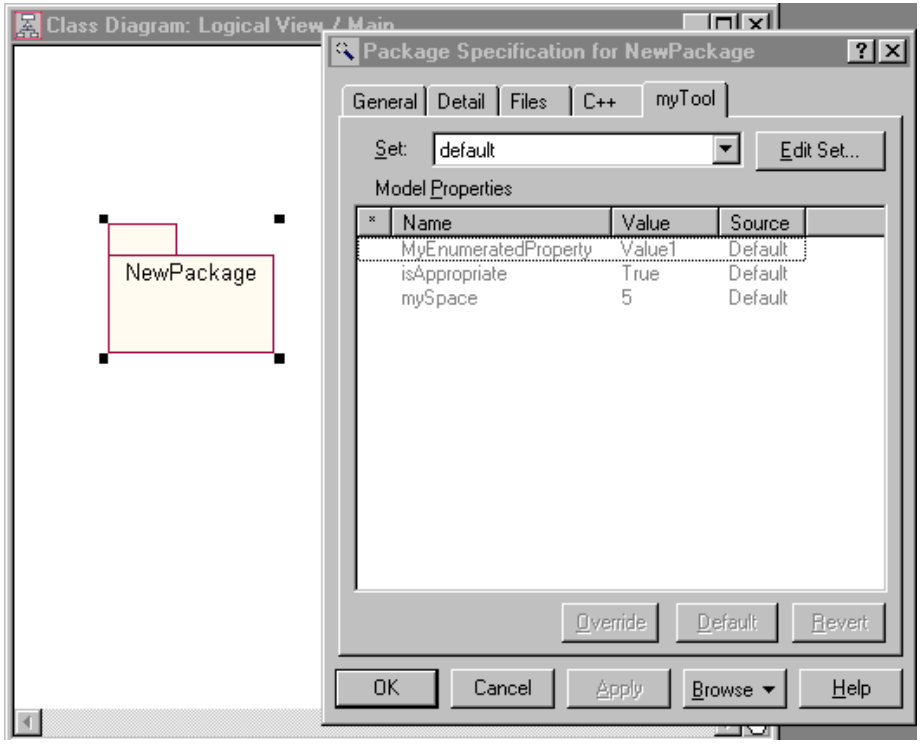
Table 15 Property File Data Types (continued)

Data type	Example and Default	Format
Integer	minCount 0	(object Attribute tool "myTool" name "minCount" value 0)
Boolean	isRelated FALSE	(object Attribute tool "myTool" name "isRelated" value FALSE)
Multi-line string	Description Blank	(object Attribute tool "myTool" name "Description" value (value Text ""))
Enumeration (setup)	Color n/a	(object Attribute tool "myTool" name "Color" value (list Attribute_Set (object Attribute tool "myTool" name "Red" value 100) (object Attribute tool "myTool" name "Blue" value 110) (object Attribute tool "myTool" name "Green" value 120))
Enumeration (usage)	Shade “Red”	(object Attribute tool "myTool" name "Shade" value ("Color" 100))

Sample Property File

The results of adding a sample property file to Rose appear in Figure 22 on page 109.

Figure 22 Sample Custom Properties



To add the tool, set, and properties (with default values) displayed in the property dialog box in Figure 22 on page 109, we created the following property file:

```
(object Petal
   version    43)

(list Attribute_Set
   (object Attribute
           tool "myTool"
           name "default__Category"
           value (list Attribute_Set
              (object Attribute
                 tool "myTool"
                 name "MyNewEnumeration"
                 value (list Attribute_Set
                    (object Attribute
                       tool "myTool"
                       name "Value1"
                       value 1)
```

```
        (object Attribute
            tool "myTool"
            name "Value2"
            value 2)
        (object Attribute
            tool "myTool"
            name "Value3"
            value 3)))
    (object Attribute
        tool "myTool"
        name "MyEnumeratedProperty"
        value ("MyNewEnumeration" 1))
    (object Attribute
        tool "myTool"
        name "isAppropriate"
        value TRUE)
    (object Attribute
        tool "myTool"
        name "mySpace"
        value 5))))
```

Note: This tool tab only appears on package specifications, since we only defined them for packages (default__Category). To display this tab for classes, duplicate the default__Category section and rename it to default__Class.

For more examples of property files, see the .pty files that come with Rose.

Creating Property Files

To create a property file, for inclusion with your add-in:

1. Create a new text file with extension .pty in a text editor or use a copy of an existing .pty file.
2. Edit the property file (.pty) as desired. For guidance, see the explanations given previously and existing property files.

Testing Property Files

To test a property file:

1. Create and save a test model with all the model elements for which you added properties.
2. Add the new property file in Rose by clicking **Tools > Model Properties > Add** and selecting your property file (.pty).
3. Check the error log to make sure your model properties were all loaded successfully. For example:

```
16:35:51|  [Add Model Properties]
```

```
16:35:51|   Adding model properties from file C:\Program
Files\Rational\Rose\my model properties.pty.

16:35:51|   A total of 4 model properties have been added to the
original model.
```

4 Test your new properties by opening the specification for each affected model element. Look for:

- A new tab or tabs with your tool name or names.
- Correct sets (default, plus any others) on each tool tab.
- Correct properties for each set.
- Correct default values for each property.
- Correct data types for each property. For example, click on an enumerated type to make sure that Rose displays a drop-down list that includes all the valid values for your enumeration.

Customizing Data Types

An add-in may also choose to supply a set of default data types to be presented to the user for typing attributes, parameters, and so on, in Rose specifications. These data types are defined in the registry setting called FundamentalTypes. For information on updating the FundamentalTypes registry setting, see *Updating the Registry* on page 123.

Customizing Stereotypes

An add-in may supply a set of stereotypes and an additional set of metafile icons to represent them. These stereotypes will be loaded and made available to Rose when the add-in is activated. Your custom stereotypes are added to Rose's default set of stereotypes for the UML. Custom stereotypes do not replace standard ones. The location of your custom stereotypes is defined in the registry setting called StereotypeCfgFile. For information on updating the StereotypeCfgFile registry setting, see *Updating the Registry* on page 123.

You may provide icons for your stereotypes or text. Stereotypes are applicable to the following model elements:

- Association
- Attribute
- Class
- Component
- Component package
- Connection

- Dependency
- Device
- Generalization
- Logical package
- Operation
- Processor
- Use case
- Use-case package

You may create custom icons for one or more of the following:

- Diagram editor icons (.wmf, .emf) to display on diagrams
- Diagram toolbar icons (.bmp) to display on the toolbar buttons
- Browser list icons (.bmp) to display in the browser

Note: You only need one bitmap file for your diagram toolbar icons and a separate bitmap file for all your custom browser icons. You do not need separate bitmap files for each of these icons. An index into the bitmap is used to indicate which bitmap goes with which stereotype.

Steps for Creating Add-In Stereotypes

To create an add-in stereotype:

1. Decide on the model element(s) and text stereotype name(s).
2. Decide which, if any, graphical representations you want to customize:
   - Editor
   - ToolBar
   - Browser
3. Define the stereotype in the stereotype .ini file for the add-in.
4. Create the custom icon graphics (.wmf, .emf, .bmp).

General .ini File Format

This section describes the .ini file format. You can also find information on custom stereotypes and the stereotype configuration file in the *Using Rose* book and online Help.

The stereotype .ini file contains the following information:

[**General**]

This section contains add-in specific settings such as the name of the add-in and whether it is a language add-in.

[**Stereotyped Items**]

This section is like a table of contents for the stereotypes. It contains a list of stereotyped REI objects. For example, Class:Control, Component:DLL, Operation:Set.

[*REI Item*:*Stereotype name*]

This section contains the settings for each stereotype, including any optional icon files and settings.

Example:

```
[General]
ConfigurationName=Name
IsLanguageConfiguration=Yes or No

[Stereotyped Items]
REI item:Stereotype name
REI item:Stereotype name
...
[REI item:Stereotype name]
Item=REI item
Stereotype=Stereotype name
optional icon settings:
Metafile=&/model-element.wmf
SmallPaletteImages=&/palette_icons.bmp
SmallPaletteIndex=Index
MediumPaletteImages=&/palette_icons.bmp
MediumPaletteIndex=Index
ListImages=&/stereotypes.bmp
ListIndex=Index
...
[REI item:Stereotype name]
Item=REI item
Stereotype=Stereotype name
optional icon settings:
Metafile=&/model-element.wmf
SmallPaletteImages=&/palette_icons.bmp
SmallPaletteIndex=Index
MediumPaletteImages=&/palette_icons.bmp
MediumPaletteIndex=Index
ListImages=&/stereotype.bmp
ListIndex=Index
```

The stereotype .ini file is composed of the following elements:

- ConfigurationName: This is the name of the add-in or name used for maintenance.
- IsLanguageConfiguration: Type **Yes** if your add-in is a language add-in. Otherwise, type **No**. This information conditionalizes stereotypes so that they only appear if the language of the model element in Rose is the same as the **ConfigurationName** listed above.

- Item: The model element for which you are defining a stereotype.
- Stereotype: The text string stereotype. This is the text to be displayed between guillemets (<< >>).
- Metafile: The windows metafile (.wmf) or enhanced metafile (.emf) containing your diagram editor icon stereotype.
  - Windows Meta Files (.wmf) may require additional extent settings.
  - Enhanced Meta Files (.emf) are preferred.
- SmallPaletteImages: The bitmap file (.bmp) containing all your small icons for your toolbar buttons. This defines non-large icons (15 pixels high x 16n wide).
- SmallPaletteIndex: The integer number indicating the location in the bitmap file of the small toolbar button icon for this stereotype. The index starts with 1.
- MediumPaletteImages: The bitmap file (.bmp) containing all your medium icons for your toolbar buttons. This defines large icons (24 pixels high x 24n wide).
- MediumPaletteIndex: The integer number indicating the location in the bitmap file of the medium toolbar button icon for this stereotype. The index starts with 1.
- ListImages: The bitmap file (.bmp) containing all your custom browser icons:
  - Device independent bitmaps
  - 16 high x 16n pixels wide
  - White background
  - Use paint or bitmap editor
  - & is the installation directory
- ListIndex: The integer number indicating the location in the bitmap file of the custom browser icon for this stereotype. The index starts with 1.

The following sections focus on the different types of text and icon stereotypes you can create. You do not need separate files for each of these items; all text and icon information can go in one .ini file.

Text-Only Stereotypes .ini File

No custom icons are included—only the text stereotypes.

Example:

```
[Stereotyped Items]
Class:Interface
Component:DLL
Component:ActiveX
```

```
Component:Application

[Class:Interface]
Item=Class
Stereotype=Interface

[Component:DLL]
Item=Component
Stereotype=DLL

[Component:ActiveX]
Item=Component
Stereotype=ActiveX

[Component:Application]
Item=Component
Stereotype=Application
```

Custom Diagram Editor Icons .ini File

`Metafile` must be used in the optional icon settings section to define diagram icons.

Example:

```
[Class:Actor]
Item=Class
Stereotype=Actor
Metafile=&/Objectory/color/actor.wmf
SmallPaletteImages=&/Objectory/palette_icons.bmp
SmallPaletteIndex=1
MediumPaletteImages=&/Objectory/palette_icons.bmp
MediumPaletteIndex=2
ListImages=&/Objectory/list_icons.bmp
ListIndex=1
```

Custom Toolbar Button Icons .ini File

`SmallPaletteImages`, **`SmallPaletteIndex`**, **`MediumPaletteImages`**, and **`MediumPaletteIndex`** must be used in the optional icons settings section to define diagram palette icons.

Example:

```
[Class:Actor]
Item=Class
Stereotype=Actor
Metafile=&/Objectory/color/actor.wmf
SmallPaletteImages=&/Objectory/palette_icons.bmp
SmallPaletteIndex=1
MediumPaletteImages=&/Objectory/palette_icons.bmp
MediumPaletteIndex=2
ListImages=&/Objectory/list_icons.bmp
ListIndex=1
```

Custom Browser List Icons .ini File

The [**General**] section is needed. **ListImages** and **ListIndex** must be used in the optional icon settings section.

Example:

```
[General]
ConfigurationName=Oracle8
IsLanguageConfiguration=Yes

[Stereotyped Items]
Class:ObjectType
Class:ObjectTable
...

[Class:ObjectType]
Item=Class
Stereotype=ObjectType
ListImages=&/o8stereo.bmp
ListIndex=3

[Class:ObjectTable]
Item=Class
Stereotype=ObjectTable
ListImages=&/o8stereo.bmp
ListIndex=4
```

Additional Online Help

Each add-in may introduce additions to the online Help when installed or activated. These additions are activated when your add-in is activated.

Online Help should cover the capabilities of the installed add-in. Each add-in should have only one first-level Help book in the master table of contents. Your add-in should add a single book, for example “My AddIn”. There may be many books under that book, but only one book should appear in the main Rose Help table of contents.

Adding Online Help for Your Add-In

To be included in the Rose Help, the add-in .hlp and .cnt files must reside in the same directory as the rest of the Rose Help.

- In Windows, the Help directory is specified by the HelpFileDir general Rose registry setting. The registry key for this setting is [HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\HelpFileDir].
- In UNIX, the Help directory is fixed: /rose.*version*/help where *version* is the version of the currently installed Rose (for example, /rose.4.5.8153/help).

In addition, you must include the add-in.cnt file in the roseu.cnt file.

To include the add-in.cnt file:

1. Locate the Rose Help files.
2. Identify the rose*X*.cnt file you wish to use:
   - Roseu.cnt supports the UML Help variant.
   - Rosec.cnt supports the COM Help variant.

   You may choose to maintain both if you refer to both the UML and COM notations when running Rose.
3. Make a backup copy of the Rose*X*.cnt file.
4. Open the rose*X*.cnt file.
5. Add the following lines to the top of the rose*X*.cnt file along with the other Index and Link definitions:

   ```
   :INDEX = title =filename.hlp

   :LINK filename.hlp
   ```

   where *title* is the text that appears in the title bar of the Contents window of the add-in Help file, *filename.hlp* is the name of the add-in Help file.
6. Add the following line at the bottom of the rose*X*.cnt file:

   ```
   :INCLUDE filename.cnt
   ```

   where *filename.cnt* is the name of the add-in's contents file.

Additional Context-sensitive Help

To be consistent with Rose, you can include context-sensitive Help in your add-in for your custom menu items, properties, and user interface (your add-in's dialog boxes, for example). Each context-sensitive Help topic must have an A-keyword defined for it. Since menu items and properties are Rose features, we explain the format needed to connect your custom menu items and properties to your context-sensitive Help. Create your A-links for your context-sensitive Help in your chosen Help authoring tool.

Main Menu Items

For main menu items, added via the menu file (.mnu), use the following format for the A-keyword.

For items on submenus:

Menu, *Submenu*, *menu item*

For items not on submenus (items located directly on Rose main menus):

Menu, *menu item*

For example, Tools, MyAddIn, MyScript would be the alias for a context-sensitive Help topic that explains the "MyScript" menu option on the "MyAddIn" submenu of the "Tools" menu (**Tools > MyAddIn > MyScript**).

Menu, submenu, and menu item names in the A-keyword must include all punctuation. For example, if your menu path includes ellipsis:

Tools > My Language > Project Specification...

Your A-keyword must also include ellipsis:

Tools, My Language, Project Specification...

Note: There is no F1 Help for intermediary submenus, only for menu items. So for the examples listed previously, there is no F1 Help for "MyAddIn" or "My Language". If you have defined Help topics and A-keywords, however, there is F1 Help for "MyScript" and "Project Specification...".

Model Properties

The format for the A-keyword is:

property (*model element*, *tool*)

where *property* is the name of your custom property, *model element* is the name of the model element to which your property is applied, and *tool* is the name of your tab (tool) in the specification.

For example, isAppropriate (Category, myTool) would be the alias for a context-sensitive Help topic that defines the "isAppropriate" property that applies to the myTool Packages.

User Manuals

You may supply your own soft or hardcopy documentation for your add-in that covers its installation, use, and limitations.

Registering for Events

You may register your add-in for Rose's events, thus triggering functionality in your add-in when that event occurs in Rose. Since any number of add-ins may trigger on the same event, the order in which your add-in entry points are called must be independent of when, where, and what add-ins are installed or activated.

Responses to events are usually coded as COM server interfaces, but some events can be mapped to Rose Scripts. Events map to either an interface on your COM server or a Rose Script, if the particular event allows Rose Script.

Some events only apply to language add-ins (for example, OnGenerateCode and OnPropertySpecOpen).

Events fall into one of the following categories:

- Registry entry required
  - Interface events
  - Script events
- No registry entry needed, but an OLE server is required

Interface and script events are explained further in the next section.

Interface vs. Script Events

Rose's registry-required events can be implemented in one of two ways, interface or script. An *interface event* requires a registered COM server (.dll) that includes an interface, named the same as the event, to handle the event. A *script event* requires a script that can be executed (.ebx) to handle the event.

What Events Are Available?

General Events:

- Model-related: OnNewModel, OnCancelModel, OnCloseModel, OnOpenModel, OnSaveModel
- Model element-related: OnNewModelElement, OnModifiedModelElement, OnModifiedModelElementEx, OnDeletedModelElement, OnDeletedModelElementEx
- When the Rose application is initialized: OnAppInit
- Add-In activation / deactivation: OnActivate, OnDeactivate

Code Generation-Related:

- Generating source code: OnGenerateCode
- Browsing source code: OnBrowseHeader, OnBrowseBody

GUI-related:

- Override the specification dialog box: OnPropertySpecOpen

- Extend the context menu: OnSelectedContextMenuItem, OnEnableContextMenuItems

Each of these events are described in detail in the online Help along with warnings and precautions. The detailed descriptions also include a *Registry and Server Requirements* section explaining whether the event requires a registry entry or OLE server.

How to Add Events to Your Add-In

This is the process for adding an event:

1. Add your COM server to your add-in registry.
2. Add events to your add-in registry.
3. Define an interface for each event.
4. Register your COM server with the operating system.

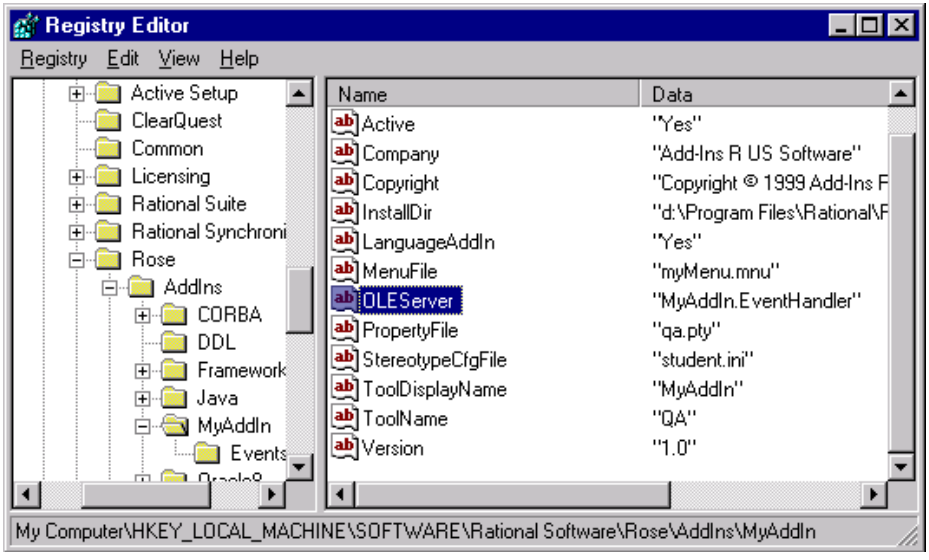
Each of these steps is detailed in the next sections.

Step 1—Adding your COM server to the add-in registry

This step is optional for Rose Script responses.

Set the OLEServer registry value to the name of your COM object (for example, MyAddIn.EventHandler).

Figure 23 OLEServer Windows Registry Entry



Step 2—Adding events to the add-in registry

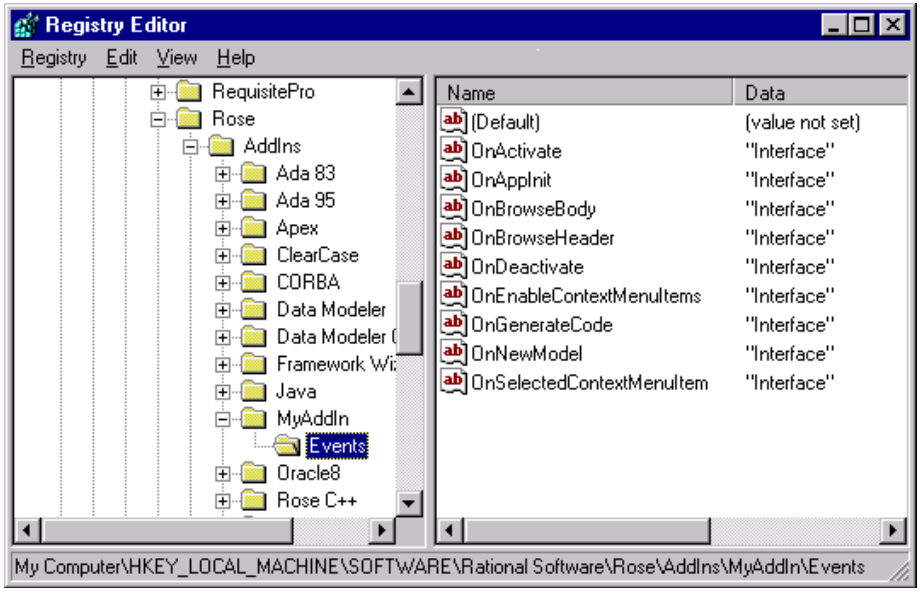
1 Create an Events registry subkey under your add-in's subkey.

2 In the **Name** column, list each event for which your add-in is registering.

3 Set the **Data** column to one of these values:

- "Interface" indicates a COM server call
- "*eventName.ebx*" indicates Rose Script execution where *eventName.ebx* is the name of your compiled RoseScript.

Figure 24 Windows Registry Entries for Rose Events



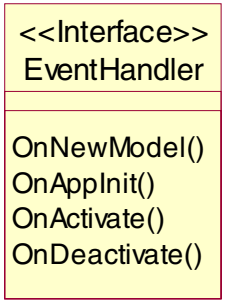
Note: The script file must reside in the add-in's installation directory, as specified by the add-in's InstallDir registry setting, or a subdirectory of the add-in's installation directory. If you choose to put the script in a subdirectory of your add-in's installation directory (for example, \scripts), specify the subdirectory as part of the script file name (for example, \scripts\OnNewModel.ebx).

Step 3—Defining an interface for each event

For each event for which your add-in is registering, name your interface or Rose Script the same. For example, if your add-in is registering for the OnNewModel event, you would have one of the following:

- OnNewModel() interface in your OLE server (see the example in Figure 25 on page 122)

Figure 25 Sample Add-In Event Handler



Note: Your COM server should only contain those events that you are responding to in your add-in.

- OnNewModel.ebx compiled Rose Script

While the signature of the interface varies by event, most interface signatures are:

```
void event_name (LPDispatch pRoseApp)
```

Step 4—Register your COM server with the operating system

Add your COM server to the windows registry so that a client can get to it by COM object name (for example, CreateObject / GetObject). This is usually taken care of by the Integrated Development Environment (IDE). For example, Visual Basic registers your .dll file for you. Otherwise, to register your COM server, execute the command line:

```
regsvr32 file.dll
```

To verify that your COM server is registered, add a reference to it in the object browser of your IDE.

Updating the Registry

Once the add-in is created, the following registry settings are necessary to enable an add-in. They are placed as subkeys to a subkey that represents the add-in name. The following would be an example of an add-in named MyAddIn:

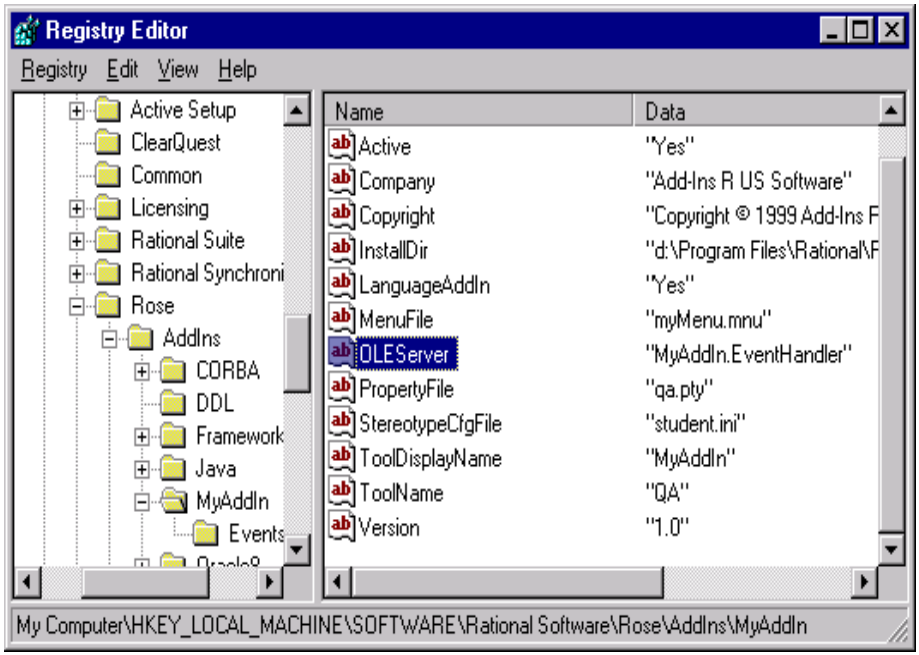
[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\AddIns\MyAddIn]

The add-in registry information should be placed in the Rose\AddIns folder of the registry.

Registry Entries

The following registry entries are available when introducing add-ins.

Figure 26 Windows Registry Entries for an Add-In



This list shows the registry subkey names, descriptions, and defaults:

Active: Whether the add-in is active or not. Can be set by the user through the Rose Add-in Manager. Default set to “Yes”.

Company: Name of the independent software vendor (ISV) that produced the add-in. For example, “Custom Software, Inc.”

Copyright: Specifies the copyright date of the add-in. For example, “©2000-2001”.

FundamentalTypes: A string list of data types that appear in the drop-down list for attributes when the add-in is active. This setting is required for all language add-ins. This field is case sensitive. For example, “LOGICAL;CHAR;REAL”.

HelpFileName: Name of the Help file for the add-in, without any path or extension. For example, “myOnlineHelp”

Note: All add-in Help files, including .cnt files, need to be located in the Help directory specified by the HelpFileDir general Rose registry setting ([HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose]).

InstallDir: Directory where the add-in is installed. For example, “d:\My AddIn”

LanguageAddIn: Whether the add-in is a round-trip engineering (RTE) language add-in that wishes to use the component mapping feature. For example, “Yes”

MenuFile: Name of the menu file (*.mnu) that tailors Rose. It needs to be installed in InstallDir. For example, “anaddin.mnu”

OLEServer: The name of the object that represents the OLE server that Rose communicates with, if the add-in uses an OLE server. For example, “MyAddIn.EventHandler”

Note: The OLEServer value is case sensitive and only required if the add-in is using an OLE server to handle events.

PropertyFile: Name of the property file (.pty) for the add-in (for example,“user.pty”). This needs to be installed in InstallDir. This registry setting is required if the add-in is introducing properties.

StereotypeCfgFile: Specifies the custom stereotype configuration file for the add-in (for example, “stereotypes.ini”). This setting is required if the add-in is introducing stereotypes. This needs to be installed in InstallDir.

ToolDisplayName: Specifies the add-in’s tool name that gets displayed on the properties tab and in the drop-down list of languages in Rose. This name can be different from the name that is used in the .pty file. Note that this is not a required setting. If this setting is not specified, the ToolName is displayed on the properties tab and in the drop-down list of languages in Rose. If this setting is specified, this is the name that gets assigned to a component. For example, “myLang” is the ToolName for the add-in, but “My Proprietary Language” is the ToolDisplayName.

ToolList: Displays the list of additional tools or property pages introduced by the add-in. Each tool is separated by a semicolon. For example, “myLang;Tool2”. This setting is only required if the add-in introduces more than one property page.

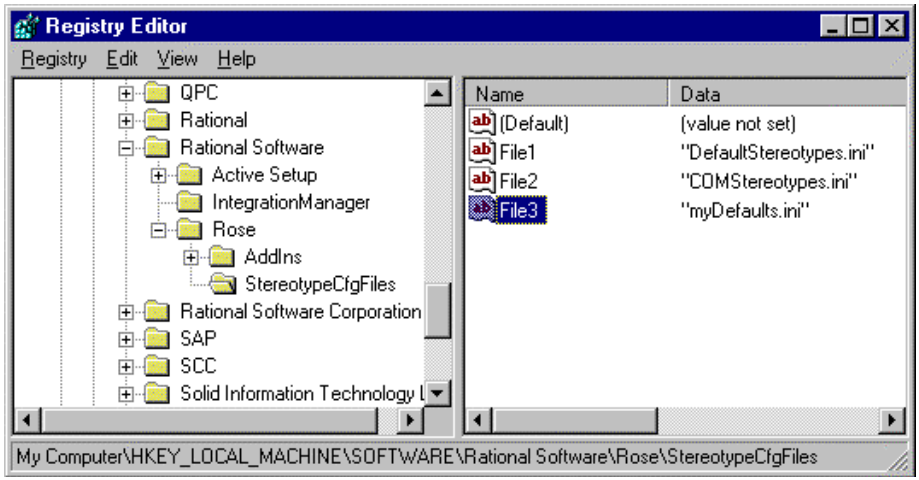
ToolName: Specifies the add-in's tool name, which must match the tool name in the add-in's .pty file (the name that gets assigned to a component). For example, "myLang", unless it is overridden by a ToolDisplayName. In that case, the ToolDisplayName is assigned to the component.

Version: Version number of the add-in, (not Rose). For example, "1.2.3"

Registering Custom Stereotypes

Add your stereotype .ini file to the StereotypeCfgFiles subkey under the Rose Subkey. Name your entry File*X* where *X* is the next available integer.

Figure 27 Registry Entry for a Custom Stereotype Configuration File



The stereotype configuration file (.ini) must be located in the directory listed in the InstallDir registry setting.

Updating the Registry During Installation

Since manual updates during installation are error-prone, we recommend that you avoid manual updates, like regedit. Instead, we suggest that you use an installation utility or execute a custom registry file.

Installation utilities

Most installation utilities (for example, InstallShield) provide programmatic interfaces to the registry. Follow your installation utility's directions for updating the registry.

Executing registry files

You can also update the registry by creating a registry file (.reg), then executing it during the installation of your add-in.

To create a custom registry file, do one of the following:

- Create a registry file (.reg) from scratch in a text editor such as Notepad following the traditional .ini file format.
- Copy and edit an existing registry file (.reg).
- Manually create your registry entries (in regedit, for example) then reverse engineer the format into a registry file (.reg):
  - Select existing add-in registry settings in a registry editor.
  - Select the menu option to export the registry.

Registry File Anatomy

A registry file (.reg) looks like the following:

```
REGEDIT4

[HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\AddIns\MyAddIn]
"Active"="Yes"
"Company"="Add-Ins R US Software"
"Copyright"="Copyright © 1999 Add-Ins R US Software Corp."
"LanguageAddIn"="Yes"
"Version"="1.0"
"PropertyFile"="qa.pty"
"MenuFile"="myMenu.mnu"
"StereotypeCfgFile"="student.ini"
"OLEServer"="MyAddIn.EventHandler"
"InstallDir"="d:\\ProgramFiles\\Rational\\Rose\\My AddIn"
"ToolName"="QA"
"ToolDisplayName"="MyAddIn"
...
```

Installing, Setting Up, and Uninstalling Your Add-In

After you finish designing and coding your add-in, it will consist of a combination of the following:

- Main menu items (.mnu)
- Shortcut menu items
- Custom specifications
- Properties (.pty)
- Data types
- Stereotypes (.ini, .bmp, .wmf, .emf)
- Online Help (.hlp, .cnt)
- Context-sensitive Help (.hlp)
- Event handling (.dll)
- Functionality through Rose Scripts (.ebx) or controls (OLE-server) (.dll, .exe)
- Installation script
- Uninstall script

The purpose of the last two items, installation and uninstall scripts, is to introduce the files into the Rose file structure and to register their locations, as well as other data needed by the framework, and to undo all this at a later time when the add-in is not wanted.

Installation Reminders

When creating your installation script, remember to do the following:

- Install the pieces of your add-in (menu file, property file, and so on) in the subdirectory indicated in your add-ins InstallDir registry subkey.
- Update the rose*X*.cnt file as needed and install your Help (.hlp) and contents (.cnt) file in the same directory as the Rose Help files.
- Update the windows registry, using your chosen method. When you update the registry, do the following:
  - Create a registry subkey for your add-in (for example, [HKEY_LOCAL_MACHINE\SOFTWARE\Rational Software\Rose\AddIns\MyAddIn])
  - Populate this subkey with the appropriate names and values (for example, InstallDir, MenuFile)
  - If using events, create an **Events** subkey under your add-in subkey. Populate the **Events** subkey with event names and “Interface” or “EventName.ebx” values.
  - If using custom stereotypes, add your stereotype configuration file name (.ini) to the **StereotypeCfgFile** subkey.
  - Create any other subkeys and registry entries needed for your implementation.

Installing Add-Ins

It is possible for an add-in supplier to provide a programmatic and complete install of the add-in, for example, by using InstallShield. The same applies to reinstalls and updates. Installation changes will not take effect while Rose is running, but will take effect upon start-up of Rose.

To install an add-in on your Windows 95, Windows 98, or Windows NT system:

1. Exit Rose.
2. Insert the application's CD-ROM or other media and run the setup.exe program.
3. Respond to the installation program's dialog boxes to complete your installation.
4. Restart Rose. Confirm that your add-in is installed and activated (checked) using the Rose **Add-In Manager** menu.

Uninstalling Add-Ins

We recommend that you provide a programmatic and complete uninstall of your add-in. Uninstall must remove not only the scripts, menu files, properties files, and Help files, but must also clean the registry entries for the add-in.

Activating and Deactivating Add-Ins

Once an add-in is installed, it can be in an activated or deactivated state. Immediately after installation, new add-ins start out as activated.

When deactivated, an add-in is all but uninstalled:

- All menu items added by the add-in are removed.
- All property tabs added by the add-in disappear.
- All event bindings added by the add-in are disengaged.

Note: A user may want to deactivate an add-in for a short time to keep it from functioning without actually uninstalling it.

Add-ins are activated and deactivated in the Rose user interface with the Add-In Manager. Add-ins are activated and deactivated programmatically with the REI AddInManager class.

Index

Symbols

106
$SCRIPT_PATH 16
%all_units variable 12
%current_diagram variable 12
%false variable 12
%model variable 12
%selected_items variable 12
%selected_units variable 12
%true variable 12
%ufile variable 12
%uname variable 12

A

accelerator key, assigning 73
accessing collections
    by count 51
    by name 52
    by unique id 51
activating, add-ins 128
Active registry entry 123
AddDefaultProperty method 42
Add-In Manager 4
adding
    comments to scripts 60
    controls 79
    menu entries 9
    property to a property set 39
    tools 48
    virtual path for scripts 16
add-ins
    activating 128
    active 123
    A-Keywords 117
    architecture 96
    basic 97
    COM servers 120
company name 123
contacting support 95
contents 97
context-sensitive help 99, 117
copyright 123
creating for UNIX 100
creating portable 100
data types 98, 111, 124
deactivating 128
developing 95, 101
events 99, 118, 119, 120, 121, 122
F1 117
functionality 99
help file name 124
inactive 123
installation directory 124
installation utilities 125
installing 125, 126, 127, 128
interface events 119
interfaces 122
language 97, 124
main menus 97, 102
manuals 118
menu file name 124
OLE server 124
online help 98, 116
portable 100
properties 98, 103, 124
property files 104, 105, 124
Rational Unified Solutions Partner Program 95
registering COM servers 122
registry 123
script events 119
setting up 126
shortcut menus 19, 98, 103
specifications 98, 103
stereotypes 98, 111, 112, 124
support 95

technical support 95
tool list 124
tools 124, 125
training 95
types 97
uninstalling 126, 128
UNIX 99
updating registry 125
user manuals 118
version 125
why create 96
adjusting attributes 83, 84
adjusting grid 71
A-Keywords
add-ins 117
properties 118
allfiles modifier 12
application object 36
assigning accelerator keys 73
Associating Files and URLs with Classes 37
Attribute 105
Attribute_Set 105
attributes, adjusting 83, 84
automation controller 4
automation server 4
Automation, Rose 4

B

basename modifier 12
basic add-ins 97
BasicScript language 3
Block Action 10
.bmp 112
Boolean 47
Braces 19
breakpoints
deleting manually 66
setting and removing 64
Browse Code 26
browser icons 111, 114, 116
adding 98

C

Calls dialog box 63
capturing dialog boxes 73
Category.AddClass method 48
Category.RelocateClass method 49
changing
titles and labels 73
value of watch variable 68
write protection for a controllable unit 8
Char 47
Classes in Categories 48
clipboard, pasting from 59
CloneDefaultPropertySet method 44
Cloning a property set 44
codefile modifier 13
collections
accessing by count 51
accessing by name 52
accessing by unique id 51
getting element from 51
COM servers 11, 118
registering 120, 122
comments
adding to script 60
property files 106
company name, add-ins 123
Company registry entry 123
compiling scripts 69
configuration files, stereotypes 112
ConfigurationName 113
context menus
See shortcut menus
context-sensitive help
add-ins 99, 117
main menus 117
properties 118
controllable units, working with 49
ControllableUnit.Control method 49
ControllableUnit.Load method 49
ControllableUnit.Save 50
ControllableUnit.SaveAs 50
ControllableUnit.Uncontrol method 49
ControllableUnit.Unload method 49
controller, automation 4

controls
adding 79
duplicating 80
incorporating in script 75
pasting into editor 82
repositioning 78
selecting 76
copying text 59
Copyright registry entry 123
copyright, add-ins 123
CreateDefaultPropertySet method 43
CreateProperty method 40, 42
creating
add-ins 96
custom specifications 103
new default property sets 38
new property 40
new property set 43
new property types 38
new scripts
from existing script 55
from scratch 55
new tools 38
portable add-ins 100
tool 48
UNIX add-ins 100
user-defined property type 47
current property set, getting and setting 46
Customized menus, capabilities of 8
customizing properties 103
customizing Rose menus, procedure 8
cutting text 59

D

data types
add-ins 98, 111, 124
customizing 98, 111
properties 107
property files 107
deactivating, add-ins 128
debugging a script 64, 65
default properties 5
DefaultModelProperties object 38
DeleteDefaultProperty method 41
DeleteDefaultPropertySet method 45
deleting
breakpoints manually 66
property 41
property set 45
text 59
watch variables 68
developing add-ins 95, 101
diagram editor icons 111, 115
adding 98
diagram toolbar icons 111
Diagram.GetSelectedItems method 50
Diagram.GetViewFrom method 50
Diagram.Items method 50
Diagram.ItemViews method 50
Diagrams 3
dialog box
capturing 73
editing 71
incorporating in script 75
inserting in script 70
pasting into editor 81
repositioning 77
selecting 76
testing 74
dialog editor, working with 70
dialogs
adding pictures to 80, 81
displaying information about 82
directories
add-ins 124
installation 124
directory modifier 13
displaying
dialog for user input 8
grid 71
information about dialogs 82
duplicating controls 80

E

editing dialogs 71
elide modifier 13

.emf 112
empty modifier 13
Enumeration 47
events
adding to your add-in 120
add-ins 99, 118
available 119
COM 99
defining interfaces for 122
details 120
interface 119, 121
main menus 11
OnActivate 119
OnAppInit 119
OnBrowseBody 119
OnBrowseHeader 119
OnCancelModel 119
OnCloseModel 119
OnDeactivate 119
OnDeletedModelElement 119
OnDeletedModelElementEx 119
OnEnableContextMenuItems 120
OnGenerateCode 119
OnModifiedModelElement 119
OnModifiedModelElementEx 119
OnNewModel 119
OnNewModelElement 119
OnOpenModel 119
OnPropertySpecOpen 26, 119
OnSaveModel 119
OnSelectedContextMenuItem 120
registering 118, 121
registering COM servers 122
script 119, 121
Exec Action 10
executing
program or shell script 8
registry files 126
Rose script 8
Extending Rational Rose course 95
extending Rose, ways to 1
Extensibility Components 2

F

F1, add-ins 117
false modifier 13
file modifier 13
file names xvi
Files, associating with Classes 37
finding
first itemview 50
procedure calls 63
text 60
FindProperty method 41
first modifier 13
Float 47
fonts, for scripts 55
FundamentalTypes registry entry 111, 124

G

Generate Code 26
GetCurrentPropertySetName method 46
GetPropertyValue method 41
getting
current property set 46
element from collection 51
model properties 41
Rose Application object
Automation 36
Scripting 36
grid, displaying and adjusting 71

H

headerfile modifier 13
help
context-sensitive 99
online 98
help file name, add-ins 124
HelpFileName registry entry 124
home_unit modifier 14

I

icons

adding 98, 111
stereotypes 111
InheritProperty method 40, 41
inserting dialog in script 70
insertion point, moving 56
installation directories, add-ins 124
installation utilities 125
InstallDir registry entry 124
installing, add-ins 126, 127, 128
Integer 47
interface events 119, 121
InterfaceEvent action 11
interfaces 118
accessing 11
defining for events 122
interscript calls
debugging 70
guidelines for 70
IsLanguageConfiguration 113
Items 50
itemview
currently selected 50
finding first 50
ItemView.IsSelected method 50
ItemViews 50
iterating
through item views 50
through the items 50

L

labels, changing 73
language add-ins 21, 97
registry entry 124
LanguageAddIn registry entry 124
language-dependent 20
language-neutral 21
ListImages 114
ListIndex 114
Load or save controllable units 8
Logical View of REI Model 2

M

main menus
add-ins 102
COM interfaces 11
context-sensitive help 117
customizing and extending 7, 97, 102
events 11
managing default properties 37
manuals, add-ins 118
MediumPaletteImages 114
MediumPaletteIndex 114
Menu Action 10, 18
Menu Argument 18
Menu Entry 18
menu files
Keywords 9
Modifiers 12
name 124
registry entry 124
syntax rules 14
Variables 12
MenuFile registry entry 124
menus
See also main menus
See also shortcut menus
Metafile 114
.mnu 102
model elements 3
property files 106
stereotypes 114
modifying value of watch variable 68
multiple modifier 14
multiple sets 5

N

non-language add-ins 22, 97
not modifier 14

O

OLEServer registry entry 120, 124
OnActivate event 119

OnAppInit event 119
OnBrowseBody event 119
OnBrowseHeader event 119
OnCancelModel event 119
OnCloseModel event 119
OnDeactivate event 119
OnDeletedModelElement event 119
OnDeletedModelElementEx event 119
OnEnableContextMenuItems event 120
OnGenerateCode event 119
online help
  adding 98, 116
  add-ins 116
OnModifiedModelElement event 119
OnModifiedModelElementEx event 119
OnNewModel event 119
OnNewModelElement event 119
OnOpenModel event 119
OnPropertySpecOpen event 26, 103, 119
OnSaveModel event 119
OnSelectedContextMenuItem event 120
opening a script 54
Option Entry 18
Option Keyword 10
OverrideProperty method 40, 41

P

pasting
  controls into editor 82
  dialog box into editor 81
  text from clipboard 59
pausing an executing script 62
petal version number, property files 106
pictures, adding to dialog 80, 81
portable add-ins 100
prefix, Rose 49
procedure calls, finding 63
properties
  adding to a property set 39
  add-ins 103
  A-Keywords 118
  context-sensitive help 118
  creating 40
  customizing 98, 103
  data types 107
  default 5
  defined 5
  deleting 41
  getting model 41
  in add-ins 98
  registry entry 124
  sets 103
  setting 41
  tool display name 124
  tool name 125
  tool registry entry 124
  tools 103
  tools list 124
  types 47
property files 5
  comments 106
  creating 110
  customizing 103
  data types 107
  design considerations 103
  format 104, 105
  model elements 106
  petal version number 106
  property name 107
  registry entry 124
  sample 109
  sets 106
  testing 110
  tools 106
property name, property files 107
property sets
  cloning 44
  creating 43
  deleting 45
  multiple 5
Property Specification Editor 38
PropertyFile registry entry 124
.pty 103, 105

R

Rational Rose Add-In Manager 4

Rational Rose Application 2
Rational Rose Application Components 2
Rational Rose Automation 3, 4
  syntax for 6
  type libraries for 49
Rational Rose diagrams, working with 50
Rational Rose Extensibility Interface 2
Rational Rose Extensibility Interface (REI) Model 1
Rational Unified Solutions Partner Program 95
.reg 126
regedit 126
registering for events 99, 118
registry
  installing 126
  updating 123, 125
registry entries 118
  Active 123
  Company 123
  Copyright 123
  FundamentalTypes 124
  HelpFileName 124
  InstallDir 124
  LanguageAddIn 124
  MenuFile 124
  OLEServer 124
  PropertyFile 124
  StereotypeCfgFile 124
  ToolDisplayName 124
  ToolList 124
  ToolName 125
  Version 125
registry files
  executing 126
  format 126
registry settings, exporting 126
regsvr32 122
REI Model
  description 1
  Logical View 2
removing breakpoints 66
replacing text 61
repositioning
  controls 78
  dialog boxes 77
  items 77
resizing items 78, 79
retrieving
  all selected items 50
  model properties 41
Rose menu file, sample 17
Rose Menus, Customizing 7
Rose prefix 49
Rose Script 3
Rose script editor 3
Rose Scripting language 3
Rose Scripts 118
Roseload Action 11
Rosesave Action 11
Rosescript Action 10
running a script 62

S

sample
  property file 109
  Rose menu file 17
  scripts 4
Script Editor
  application window 54
  selecting a font 55
script events 119, 121
Scripting language 3
scripts
  sample 4
  virtual path for 16
selecting
  control 76
  dialog boxes 76
  fonts 55
  text 57, 58
  variables 68
Separator Entry 18
Separator Keyword 10
server, automation 4
SetCurrentPropertySetName method 46
sets
  properties 103
  property files 106

setting
current property set 46
model properties 41
Using InheritProperty 43
Using OverrideProperty 42
setting up, add-ins 126
shortcut menu items
adding 27
changing states 28
displaying 23
editing 28
formatting and displaying 22
retrieving 27
shortcut menus
activities 21
add-ins 98, 103
benefits 19
customizing 19, 26, 98, 103
decisions 21
deployment units 21
designing 26
diagrams 21
extending 7, 98
external documents 21
formatting 23
how it works 22
language-neutral model elements 21
limitations 20
menu items
adding 27
changing states 28
editing 28
formatting and displaying 22
retrieving 27
models 21
packages 21
properties 21
sample scripts 28, 30
scenarios 23
states 21
subsystems 21
swimlanes 21
synchronizations 21
transitions 21
use cases 21
SmallPaletteImages 114
SmallPaletteIndex 114
sourcefile modifier 14
specifications
add-ins 98, 103
custom, opening 26
customizing 98, 103
standard, opening 26
StepInto tool 63
StepOver tool 63
Stereotype 114
StereotypeCfgFile registry entry 111, 124
StereotypeCfgFiles 125
stereotypes
add-in type 113
add-ins 98
bitmap files 114
browser icons 114, 116
configuration file format 112
configuration name 113
creating 112
customizing 98, 111
diagram editor icons 114, 115
enhanced metafile 114
icons 98
model elements 114
registering 125
registry entry 124
text string 114
text-only 114
toolbar button icons 114, 115
windows metafile 114
stopping an executing script 62
String 47
Submenu Entry 18
Summit BasicScript language 3
syntax
in Rose Automation 6
in Rose Script 6
REI 4
syntax rules, menu file 14

T

testing dialog boxes 74
text-only stereotypes 114
titles, changing 73
Tool, defined 5
toolbar button icons 114, 115
  adding 98
  stereotypes 114
ToolDisplayName registry entry 124
ToolList registry entry 124
ToolName registry entry 125
tools
  display name 124
  list 124
  name 125
  properties 103
  property files 106
  registry entry 124, 125
tracing script execution 63
tracking variables 66
type libraries 5, 49
Type, defined 5
type, property 47

U

unary modifier 14
uninstalling, add-ins 126, 128
UNIX file names xvi
UNIX versus Windows, add-ins 99
Updateaccess Action 11
URL, associating with Classes 37
user manuals, add-ins 118
using type libraries 49
utilities, installation 125

V

Variables 12
variables with modifiers, syntax 11
variables, tracking 66
Version registry entry 125
virtual path for scripts 16

W

watch list 68
watch variables, adding 66
.wmf 112
working with controllable units 49
working with Rose diagrams 50
writeable modifier 14