Cisco Active Network Abstraction Command Builder User Guide Version 3.6

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-12505-01

Cisco Active Network Abstraction Command Builder User Guide, Version 3.6

CONTENTS

About This Guide vii

Obtaining Documentation, Obtaining Support, and Security Guidelines viii

CHAPTER 1 **Introducing the Command Builder** 1-1

About the Command Builder 1-1

CHAPTER 2 **Getting Started** 2-1

Opening the Command Builder 2-2

Command Builder Wizard 2-3

Menu Bar 2-4

File Menu 2-4

Tools Menu 2-5

Toolbar 2-5

A Workflow To Define a New Command 2-6

Creating or Editing a Command 2-7

Editing a Command 2-8

Defining the User Input Parameters 2-9

Defining a Combo Field Type 2-12

Defining ANA Macro Language Script Lines 2-13

Defining Bean Shell Script Lines 2-15

Previewing and Executing the Command 2-16

Publishing Commands 2-18

Deleting a Command 2-19

Importing and Exporting a Command 2-20

Reviewing the Command History 2-21

Closing the Command Builder 2-21

CHAPTER 3 **Creating an ANA Macro Language Command: An Example** 3-1

Creating a Command 3-1

CHAPTER 4 **Running Command Scripts** 4-1

Syntax 4-1

Output 4-2

Examples 4-2

ChangePortStatus 4-2

Add_VRF 4-2

RunScript 4-3

Script Results 4-3

IScriptResult 4-3

IScriptEvent 4-4

Successful Example 4-5

Unsuccessful Example 4-6

APPENDIX A **ANA Macro Language** A-1

What Are ANA Macro Language Scripts? A-1

Properties Available From the IMO Context A-2

Specifying and Using Parameters A-2

User-Defined Parameters A-3

Multiple Formats for IP Subnet Parameters A-3

Built-In Parameters A-4

Supported Pragmas A-4

Success A-5

Description A-5

Syntax A-5

Directives A-5

Examples A-5

Fail A-5

Description A-5

Syntax A-5

Directives A-6

Example A-6

Prompt A-6

Description A-6

Syntax A-6

Directives A-6

Example A-6

Full Prompt A-7

Description A-7

Syntax A-7
Directives A-7
Example A-7
Rollback A-7
Description A-7
Directives A-7
Activity A-8
Description A-8
Syntax A-8
Directives A-8
Example A-8
Enum A-8
Description A-8
Directives A-8
Example A-8
Example A-9
Command Script A-9
Rollback Script A-9
Running the Script A-10

APPENDIX B **Bean Shell Commands** B-1
Telnet Examples B-2
SNMP Examples B-2

About This Guide

This user guide describes managing command scripts. Command scripts enable the user to execute a programmable sequence of SNMP or Telnet command lines. These commands can include data properties taken from the Cisco Active Network Abstration (ANA) information model (built-in), as well as user-defined input parameters entered during runtime.

In addition, it describes the ANA Macro Language, its syntax, how to use parameters, pragmas and a detailed example for writing ANA Macro Language scripts. This guide is intended for use by programmers who want to write command scripts that are executed within the Cisco ANA activation framework.

It includes the following chapters:

- Chapter 1, “Introducing the Command Builder” describes the Command Builder tool.
- Chapter 2, “Getting Started” describes the Command Builder wizard's working environment and how to access the Command Builder tools. In addition, it describes how to create, execute, and publish a command.
- Chapter 3, “Creating an ANA Macro Language Command: An Example” describes creating an ANA Macro Language command from start to finish.
- Chapter 4, “Running Command Scripts” describes running command scripts.
- Appendix A, “ANA Macro Language” describes the ANA Macro Language, its syntax, how to use parameters, pragmas and a detailed example for writing ANA Macro Language scripts.
- Appendix B, “Bean Shell Commands” describes the methods that should be used for Bean Shell in Cisco ANA commands when you want to interact with devices. In addition, it provides Telnet and SNMP environment object examples.

Note Changes to the registry should only be carried out with the support of Cisco Professional Services.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

CHAPTER 1

Introducing the Command Builder

This user guide describes managing command scripts. Command scripts enable the user to execute a programmable sequence of SNMP or Telnet command lines. These commands can include data properties taken from the Cisco ANA information model (built-in), as well as user-defined input parameters entered during runtime.

About the Command Builder

The Command Builder tool is designed to enable the user to create new script and activation commands.

Since the Command Builder is built-in to the standard Cisco ANA system platform:

- It utilizes Cisco ANA's regular access interfaces to the network.
- Commands can be associated with any existing object group (IMO), type or instance. This is the "working object" on which the command is developed and tested.
- Enables access to the live information of the network object with which the command is associated.

The Command Builder tool enables the user to create commands with two different languages:

- ANA Macro Language—Using a GUI wizard. For information about ANA Macro Language scripts see Appendix A, "ANA Macro Language".
- Bean Shell scripting language.

Once defined, the commands are executed either:

- Interactively—Via auto-generated GUI forms (NetworkVision standard menus).
- Flow-through—Cisco ANA API, to integrate with external configuration applications.

The command can be published once it has been completed, enabling a wider scope of managed elements and network elements to use it.

Note Changes to the registry should only be carried out with the support of Cisco Professional Services.

CHAPTER 2

Getting Started

This chapter describes the Command Builder wizard's working environment and how to access the Command Builder tools. In addition, it describes how to create, execute, and publish a command.

- Opening the Command Builder, page 2-2—Describes how to open the Command Builder wizard.
- Command Builder Wizard, page 2-3—Describes the Command Builder wizard, including, the toolbar and menu options.
- A Workflow To Define a New Command, page 2-6—Describes the steps required to create a new command.
- Creating or Editing a Command, page 2-7—Describes how to start creating a command. In addition, it describes how to edit a command.
- Defining the User Input Parameters, page 2-9—Describes how to define the input parameters and properties, which will be used when the command is executed.
- Defining ANA Macro Language Script Lines, page 2-13—Describes how to define ANA Macro Language script lines.
- Defining Bean Shell Script Lines, page 2-15—Describes how to define Bean Shell script lines.
- Previewing and Executing the Command, page 2-16—Describes how to preview and/or execute a command.
- Publishing Commands, page 2-18—Describes how to publish a command to one or more locations across the inheritance hierarchy.
- Deleting a Command, page 2-19—Describes how to delete a local instance of a command.
- Importing and Exporting a Command, page 2-20—Describes how to export and import commands between managed elements.
- Reviewing the Command History, page 2-21—Describes how to view the command's execution history.
- Closing the Command Builder, page 2-21—Describes how to close the Command Builder wizard.

Opening the Command Builder

This section provides instructions for launching the Command Builder wizard. The Command Builder is launched from a specific managed element, which could be a managed element or a selected object within a managed element, such as a port. This managed element will be used to develop and test the command. Once the command has been completed it can be published and attached to a wider scope of managed elements.

To open the Command Builder:

Step 1 Right-click on a managed element in the tree pane or context pane of the NetworkVision window.

Note For more information about the NetworkVision window see the *Cisco Active Network Abstraction NetworkVision User Guide*.

or

Open the Inventory window for the required managed element and right-click on the required VNE inventory item.

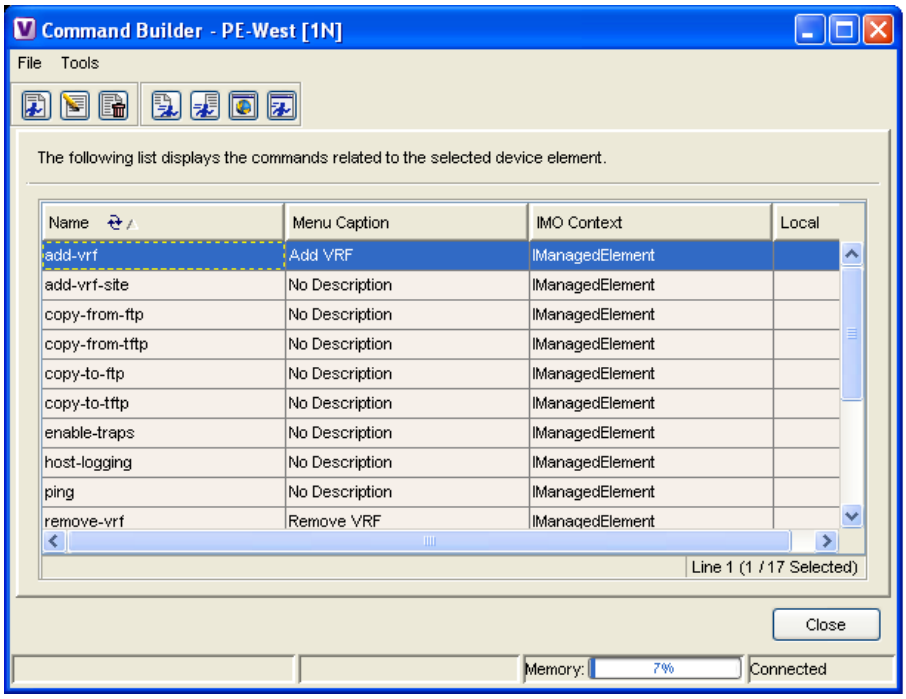
Note For more information about the Inventory window see the *Cisco Active Network Abstraction NetworkVision User Guide*.

Step 2 Select **Management | Command Builder** from the shortcut menu. The Command Builder wizard is displayed.

Command Builder Wizard

An example of the Command Builder wizard is displayed below.

Figure 2-1 Command Builder Wizard



The Command Builder wizard consists of the following:

- The table of commands for the selected managed element or network element, as described below.
- Menu Bar, page 2-4
- Toolbar, page 2-5

The Command Builder wizard displays a table of all the existing commands that are available for the selected managed element and/or network element. The Command Builder enables the user to:

- Add a new command, which enables the user to execute a programmable sequence of Telnet or SNMP command lines.
- Edit an existing command.
- Delete a command that has not been published.
- Import and/or export command definitions.
- Run or test a command on the selected managed element.
- Publish a command and attach it to a wider scope of managed elements.

The following information is displayed in the table of the Command Builder wizard:

- Name—The identifier of the command. This name is unique to the entire system.
- Menu Caption—The text that describes the command in the menu when launching the command.
- IMO Context—The inventory object with which the command is associated.

Note A command is always associated with a selected object within a managed element, which enables it to use the properties of this object inside the script lines. For example, by selecting a port object, the port's properties like portAlias, and status are automatically made available to the script.

- Local—Specifies whether the command is inherited from a higher level or is defined locally on the selected managed element. A, command that is defined for a scope of managed elements (such as "All devices" or a specific device type), is automatically assigned to all the managed elements in that scope. When modifications are made to a command that is inherited from a higher level, a local copy of the command is created for the specific managed element and overrides the generic definition. Once the local copy is tested and accepted, it can be published in order to update the higher-level definition.

A table can be sorted:

- According to a column by clicking on the required column heading. The icon is displayed next to the selected column heading.
- In ascending or descending order by clicking on the column heading. A triangle is displayed next to the selected column heading.

Clicking on a red triangle displayed in a cell expands the cell to display all the information in the cell.

The Location field displays the number of selected rows and the total number of rows in the table, for example, 2/16 Selected. In addition, it displays the location of the selected row(s) in the table, for example, Line 3.

Menu Bar

This section provides a description of each option available in the Command Builder menus. The following menus are available:

- File Menu, page 2-4
- Tools Menu, page 2-5

File Menu

The File menu provides the following options:

- New Element—Enables the user to create a new command definition. For more information see Creating or Editing a Command, page 2-7.
- Edit Element—Enables the user to edit an existing command definition. For more information see Editing a Command, page 2-8.
- Delete Element—Enables the user to delete existing locally defined commands, namely, commands that have not yet been published. For more information see Deleting a Command, page 2-19.

Tools Menu

The Tools menu provides the following options:

- Export Element—Enables the user to save a full command definition to a file that can later be imported to another managed element. For more information see Importing and Exporting a Command, page 2-20.
- Import Element—Enables the user to copy a full command definition from an exported file and import this command to another managed element. For more information see Importing and Exporting a Command, page 2-20.
- Hierarchy Manager—Enables the user to move the command definition to a different location or change the scope of the command across the network hierarchy. For more information see Publishing Commands, page 2-18.
- Run Command—Enables the user to preview and/or execute the command. For more information see Previewing and Executing the Command, page 2-16.

Toolbar

The following buttons are displayed in the Command Builder wizard:

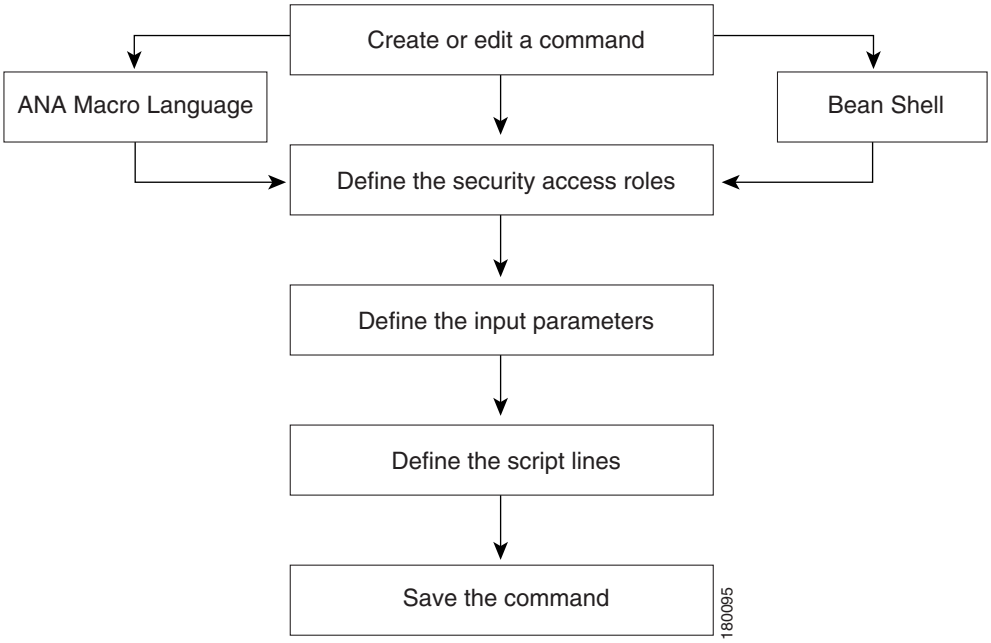
Button	Description
	**New Element—**Enables the user to create a new command definition. For more information see Creating or Editing a Command, page 2-7.
	**Edit Element—**Enables the user to edit an existing command definition. For more information see Editing a Command, page 2-8.
	**Delete Element—**Enables the user to delete commands that exist locally, namely, commands that have not yet been published. For more information see Deleting a Command, page 2-19.
	**Export Element—**Enables the user to save a full command definition to a file that can later be imported to another managed element. For more information see Importing and Exporting a Command, page 2-20.
	**Import Element—**Enables the user to copy a full command definition from an exported file and import this command to another managed element. For more information see Importing and Exporting a Command, page 2-20.
	**Hierarchy Manager:** Enables the user to move the command definition to a different location or change the scope of the command across the network hierarchy. For more information see Publishing Commands, page 2-18.
	**Run Command:** Enables the user to preview and/or execute the command. For more information see Previewing and Executing the Command, page 2-16.

The **Close** button closes the Command Builder wizard. For more information see Closing the Command Builder, page 2-21.

A Workflow To Define a New Command

The workflow below describes the steps required to define a new command definition using the Command Builder and the order in which they must be performed.

Figure 2-2 Define A New Command Workflow



At any point after the command has been defined, it can be tested, executed, and published to a wider scope of managed elements and/or network elements.

For more information about:

- Creating or editing a command see Creating or Editing a Command, page 2-7.
- Defining the security access roles see Creating or Editing a Command, page 2-7.
- Defining the input parameters see Defining the User Input Parameters, page 2-9.
- Defining the script lines and saving the command see Defining ANA Macro Language Script Lines, page 2-13 for ANA Macro Language and Defining Bean Shell Script Lines, page 2-15 for Bean Shell.
- Previewing and executing the command see Previewing and Executing the Command, page 2-16.
- Publishing the command see Publishing Commands, page 2-18.
- Deleting a command see Deleting a Command, page 2-19.
- Importing and exporting a command see Importing and Exporting a Command, page 2-20.
- Reviewing the command history see Reviewing the Command History, page 2-21.

Creating or Editing a Command

The Command Builder enables the user to create a command definition, which by default is created as a local instance. For more information about publishing a local instance of a command to a higher level in the hierarchy see Publishing Commands, page 2-18.

In addition, the user can edit an existing command. For information about editing an existing command see Editing a Command, page 2-8.

To create a command:

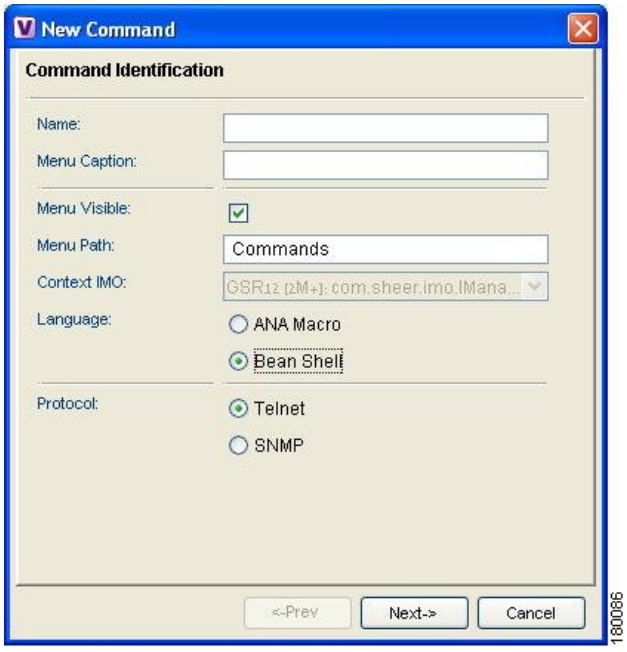
Step 1 Click **New Element** on the toolbar of the Command Builder wizard.

or

Select **New Element** from the File menu or shortcut menu.

The New Command dialog box is displayed.

Figure 2-3 New Command Dialog Box



The following information is displayed in the New Command dialog box:

- Name—The name identifying the command and this name is unique to the entire system.
- Menu Caption—The text that describes the command in the menu when launching the command, namely, the menu option.
- Menu Visible—Select this option to display the menu option in the NetworkVision shortcut menu or clear this option to hide the menu option from the user.

Note A command that is defined as not visible in the menu can still be executed via the API.

- Menu Path—The location in the menu where the command will be added or displayed.

- Context IMO—Displays the inventory object that is associated with the command (and exposes its data properties to the command). In a case where the inventory object has sub-objects, which do not appear in the inventory tree (for example, parameter-groups of a port), they will be listed in the dropdown list.
- Language—Select either ANA Macro Language (Simple mode) or Bean Shell (programmable mode/full scripting language).
- Protocol—Select the protocol:
  - ANA Macro Language uses only Telnet.
  - Bean Shell can use either Telnet or SNMP.

Note Cisco ANA provides a SNMP Bean Shell template that enables easy creation of SNMP commands.

The following buttons are displayed in the New Command dialog box:

- Next—Displays the Command Authorizations dialog box.
- Cancel—Returns to the opening window of the Command Builder wizard.

Step 2 Define the command identification information.

Step 3 Click **Next**. The Command Authorizations dialog box is displayed.

The Command Authorizations dialog box enables the user to specify which security access roles will be authorized to execute the command, namely:

- Administrator
- Configurator
- Operator Plus
- Operator
- Viewer

The following buttons are displayed in the Command Authorizations dialog box:

- Prev—Displays the New Command dialog box.
- Next—Displays the User Input Arguments dialog box.
- Cancel—Returns to the opening window of the Command Builder wizard.

Step 4 Select the required security access role/s.

Step 5 Click **Next**. The User Input Arguments dialog box is displayed.

Proceed to Defining the User Input Parameters, page 2-9 to continue creating the command.

Editing a Command

The user can edit an existing command definition and the command that is edited will affect only the local instance. When an inherited command is edited, the new local instance overrides the generic command definition for the specific managed element.

To edit a command:

Step 1 Select the command that you want to edit in the table of the Command Builder wizard.

Step 2 Click **Edit Element** on the toolbar of the Command Builder wizard.

or

Select **Edit Element** from the File menu or shortcut menu.

The Hierarchy Manager dialog box is displayed.

Note If user-friendly VNE names exist in the schema then the hierarchy manager table will display these user-friendly registry location names in the VNE Hierarchy Location column. A user-friendlyVNE name is a hierarchy path that has been defined in the registry and is then displayed in the hierarchy manager table.

Step 3 Select the required version of the command from the hierarchy manager, and click **Next**. The Edit Command dialog box is displayed for the selected command.

Step 4 Edit the command as required by changing the existing information using the Command Builder wizard.

Step 5 Click **Finish** in the Script Lines dialog box to save the edited ANA Macro Language command.

or

Click **Finish** in the Bean Shell Script dialog box save the edited Bean Shell command.

Step 6 When the command has been successfully saved click **Close**. The edited command is supported and displayed in the Command Builder wizard.

For information about previewing and executing the edited command see Previewing and Executing the Command, page 2-16.

For information about publishing the edited command see Publishing Commands, page 2-18.

Defining the User Input Parameters

The Command Builder enables the user to define any number of input parameters. The input parameter's attributes determine the structure and format of the input form. When the command is executed the input form is generated automatically.

Note The order of the input parameters determines the order in which they are presented in the input form.

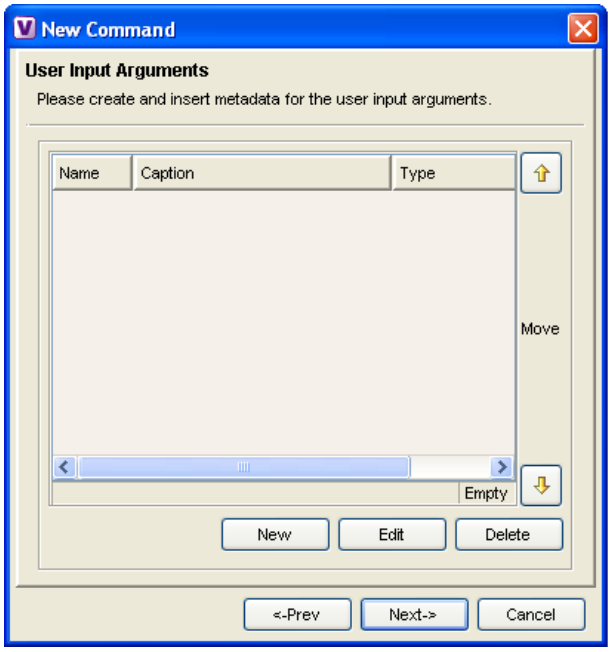
ANA Macro Language supports two types of script parameters. User-defined parameters and built-in parameters, both are replaced in runtime. All parameters (both built-in and user-defined ones) are available during command editing via a selection list.

For information about ANA Macro Language scripts see Defining ANA Macro Language Script Lines, page 2-13.

To define the input parameters:

Step 1 Click **Next** in the Command Authorizations dialog box. The User Input Arguments dialog box is displayed.

Figure 2-4 User Input Arguments Dialog Box



The following is the complete list of user-defined parameter properties that can be defined in the Add/Edit User Argument for x Command dialog box:

Property	Explanation
Name	Parameter name. Can contain only letters, numbers, "-" and "_". Unique within the script scope.
Caption	Parameter display name. Visible in the Command Builder script execution window.
Type	String, Integer, IPSubnet, Combo, IP Address, Float, Long. For more information about defining a **Combo** see Defining ANA Macro Language Script Lines, page 2-13. **Note** Only input values that are valid for the selected **Type** will be accepted. These values are validated during runtime.
Width	Field width, in characters. Relevant for the Command Builder script execution window.
Visible	Select this option to show this parameter in the Command Builder script execution window or clear this option to hide the parameter from the user. In the case where the argument is hidden, it can still be used in the command (with its default value). **Note** When the parameter is not visible and has been assigned a default value it can serve as a constant argument.
Default	A default value for the parameter.

The User Input Arguments dialog box **Move Up** and **Move Down** buttons enable the user to determine the order of the user input parameters thereby determining the order in which they are presented when they are executed.

The following additional buttons are displayed in the User Input Arguments dialog box:

- New—The Add/Edit User Argument for x Command dialog box is displayed enabling the user to create a new input argument for the command.

Note A discussion of the Advanced option does not fall within the scope of this guide.

- Edit—The Add/Edit User Argument for x Command dialog box is displayed enabling the user to edit the selected input argument.
- Delete—Deletes the selected input argument from the User Input Arguments table.
- Prev—Displays the Command Authorizations dialog box.
- Next—Displays the Script Lines dialog box.
- Cancel—Returns to the opening window of the Command Builder wizard.

Step 2 Click **New**.

or

Select the input argument and click **Edit**.

The Add/Edit User Argument for x Command dialog box is displayed.

Step 3 Enter or edit the information for the input argument.

Step 4 Click **OK**. The new or edited user input argument is displayed in the User Input Arguments dialog box.

Note Select a user-defined input parameter and click the **Move Up** or **Move Down** buttons to reorder the user-defined input parameters.

Step 5 Click **Next** to define:

- ANA Macro Language command, the Script Lines dialog box is displayed.

  See Defining ANA Macro Language Script Lines, page 2-13 to continue creating the command.

- Bean Shell command, the Bean Shell Script dialog box is displayed.

  See Defining Bean Shell Script Lines, page 2-15 to continue creating the command.

Defining a Combo Field Type

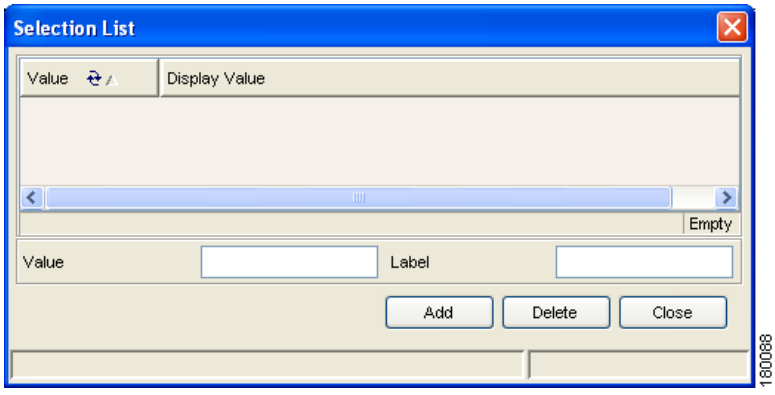
When **Combo** is selected in the **Type** field of the Add/Edit User Argument for x Command dialog box the **Browse** button is enabled. This enables the user to create a selection list (dropdown list) of the valid options that will be displayed in the combo box of the input form, for example, Up = 1 and Down = 2.

To define the combo entries:

Step 1 Select **Combo** in the **Type** field of the Add/Edit User Argument for x Command dialog box.

Step 2 Click **Browse**. The Selection List dialog box is displayed.

Figure 2-5 Selection List Dialog Box



The following information is displayed in the Selection List dialog box:

- **Value—**The actual value of the option, for example, “1”.
- **Label—**The description of the entry that is displayed in the selection list (dropdown list) of the input form, for example, “Up”.

The following buttons are displayed in the Selection List dialog box:

- **Add—**Adds a new entry to the selection list.
- **Delete—**Deletes the selected entry from the selection list.
- **Close—**Closes the Selection List dialog box. The Add/Edit User Argument dialog box is displayed.

Step 3 Enter the required **Value** and **Label** in their respective fields.

Step 4 Click **Add**. The entry (**Value** and **Label**) is added to the table.

Step 5 Repeat Step 3 and Step 4 to enter all the required entries.

Step 6 Click **Close**. The Add/Edit User Argument dialog box is displayed.

Defining ANA Macro Language Script Lines

The user can define the language of the script as either **ANA Macro Language** or **Bean Shell**. ANA Macro Language scripts are a simple sequence of Telnet commands, runtime replaced user-defined input parameters and inline execution directives that are executed sequentially as telnet configuration commands on a networking device.

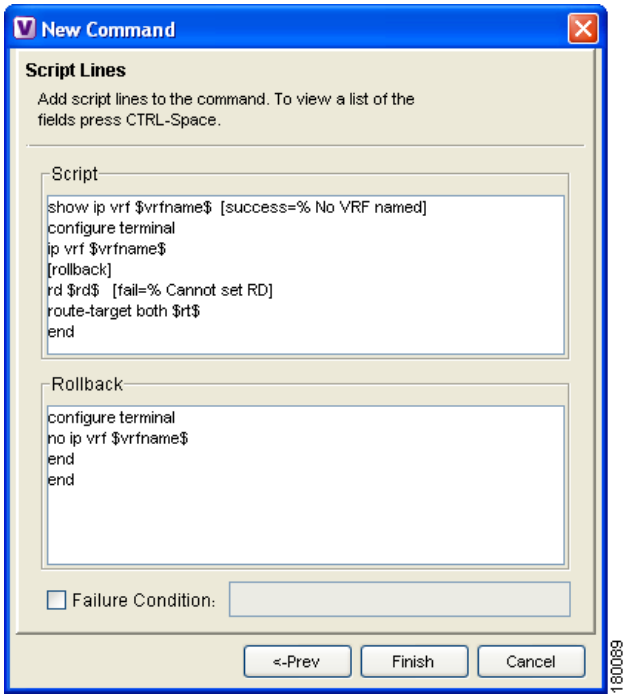
ANA Macro Language represents both types of parameters (namely, built-in and user-defined ones) in script lines within dollar symbols, namely, **$...$**. For example, in a VRF configuration command, the input variable vrfName can be defined as ip vrf $vrfName$.

For information about ANA Macro Language scripts see Defining ANA Macro Language Script Lines, page 2-13.

To define ANA Macro Language script lines:

Step 1 In the User Input Arguments dialog box click **Next**. The Script Lines dialog box is displayed enabling the user to add or edit a script line, as displayed below.

Figure 2-6 Script Lines Dialog Box



Note To view all the user-defined and built-in parameters in the Command Builder application press CTRL + Space to open the selection list of available arguments (containing both the user-defined input argument and the built-in properties of the IMO context).

The following areas are displayed in the Script Lines dialog box:

- Script—The actual Telnet script lines sent to the device. The script lines can contain optional inline directives (pragmas) for finer-granularity control. For more information about the supported pragmas see Supported Pragmas, page A-4.

- Rollback—The rollback script that will be used when the command fails (optional).

Note If the rollback script fails, no additional actions can be performed.

- Failure Condition—A general failure condition, which applies to all the script lines (optional). Aborts the command if the specified text exists in the reply.

The following buttons are displayed in the Script Lines dialog box:

- Prev—Displays the User Input Arguments dialog box.
- Finish—Saves the command to the registry and the Create Command dialog box is displayed while the command is being created.
- Cancel—Returns to the opening window of the Command Builder wizard.

Step 2 Define the script lines in the **Script** area.

Note Pragmas are enclosed with square brackets, namely, […].

Note To view all the user-defined and built-in parameters in the Command Builder application press CTRL + Space to open the selection list of available arguments (containing both the user-defined input argument and the built-in properties of the IMO context).

Note Wherever carriage returns are required in the command line the programmer must use &cr.

Note It is possible to use multiple pragmas in a single line, in which case all the pragmas will be analyzed. If the same type of pragma is repeated only the last one will be used.

Step 3 Define the **Rollback** and **Fail Condition** criterion (optional).

Step 4 Click **Finish**. The Create Command dialog box is displayed.

LEDs indicate the progress or status of the command as it is being saved to the registry as follows:

- Blue—The command definitions are now being saved.
- Green—The command has been created/updated successfully.
- Red—The Command Builder wizard failed to create/update the command.

Step 5 Click **Close** when the command has been successfully saved. The newly created command is displayed in the Command Builder table.

See Previewing and Executing the Command, page 2-16 to preview and/or execute the command.

See Publishing Commands, page 2-18 to publish the command.

Defining Bean Shell Script Lines

The user can define the language of the script as either ANA Macro Language or Bean Shell. Bean Shell uses a fully programmatic logic via scripting language (conditions, loops, external files, etc.).

For information about ANA Macro Language scripts see Defining ANA Macro Language Script Lines, page 2-13.

To define Bean Shell Script Lines:

Step 1 In the User Input Arguments dialog box click **Next**. The Bean Shell Script dialog box is displayed.

Note To view all the user-defined and built-in parameters in the Command Builder application press CTRL + Space to open the selection list of available arguments (containing both the user-defined input argument and the built-in properties of the IMO context).

Note Important: Unlike ANA Macro Language, in Bean Shell user arguments inventory properties should NOT be embedded within $...$ symbols.

The following areas are displayed in the Bean Shell Script dialog box:

- Script—The actual Telnet script lines sent to the device. The script lines can contain optional inline directives (pragmas) for finer-granularity control.
- Rollback—The rollback script that will be used when the command fails (optional).

The following buttons are displayed in the Bean Shell Script dialog box:

- Prev—Displays the User Input Arguments dialog box.
- Finish—Saves the command to the registry and the Create Command dialog box is displayed.
- Cancel—Returns to the opening window of the Command Builder wizard.

Step 2 Press **Ctrl- Spacebar** and select the required field.

Step 3 Click **OK**. The selected field is displayed in the Bean Shell Script dialog box.

Step 4 Repeat Step 2 and Step 3 to add all the required fields.

Step 5 Click **Finish**. The Create Command dialog box is displayed. LEDs indicate the progress or status of the command as it is being saved to the registry. For more information see Defining ANA Macro Language Script Lines, page 2-13.

Step 6 Click **Close** when the command has been successfully saved. The newly created command is displayed in the Command Builder table.

See Previewing and Executing the Command, page 2-16 to preview and/or execute the command.

See Publishing Commands, page 2-18 to publish the command.

Previewing and Executing the Command

Command Builder enables the user to preview the command (including the variables) before it is executed. The input form is automatically generated and opened displaying all the user-defined input fields for the command. The Command Builder supports multiple activations, meaning that a command can be invoked to run concurrently on multiple managed elements or network elements.

To preview or execute the command:

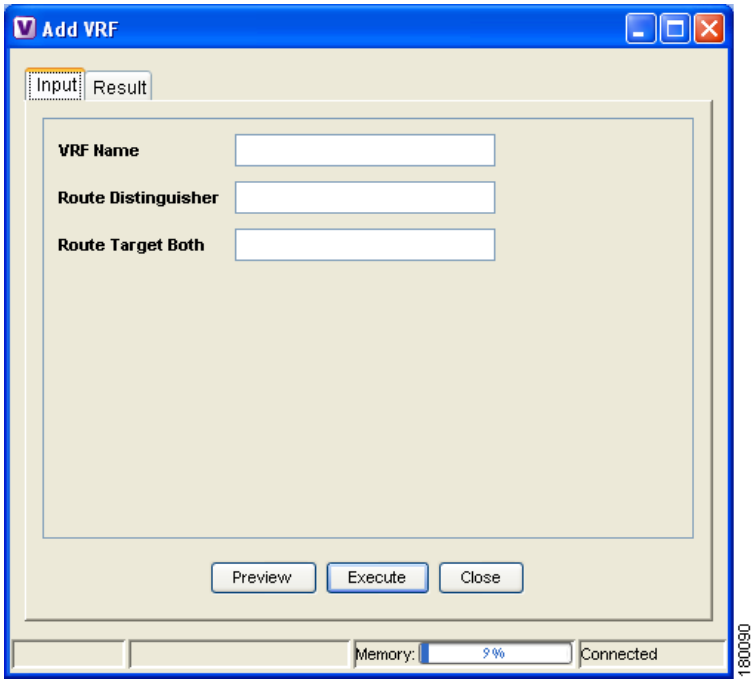
Step 1 Select the required command in the Command Builder wizard table.

Step 2 Click **Run Command** on the toolbar of the Command Builder wizard.

or

Select **Run Command** from the Tools menu or shortcut menu. The input form is generated and displayed.

Figure 2-7 Input Form



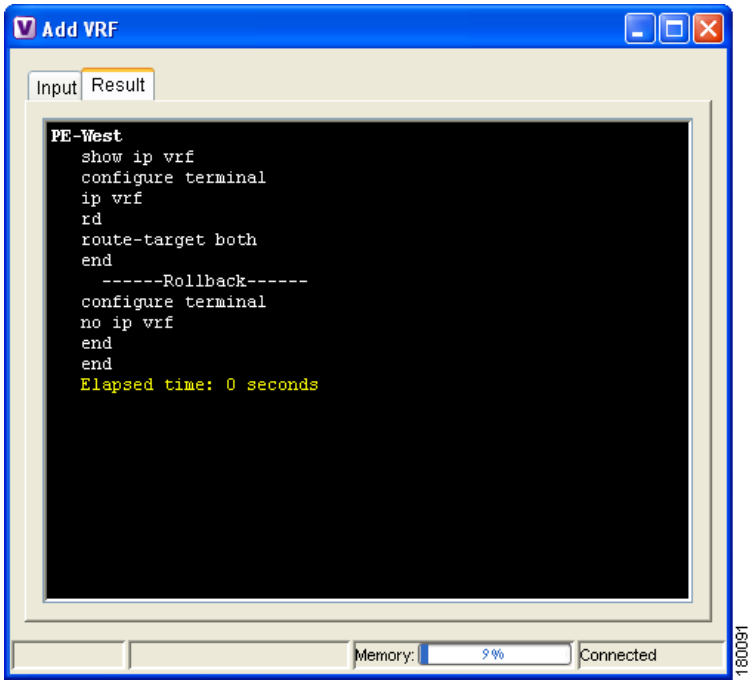
The input form heading displays the name of the command and is comprised of two tabs:

- The Input tab—Displays the input parameters.
- The Result tab—Displays the preview or the actual interaction of the command in the output console with full execution audit.

The following buttons are displayed in the input form:

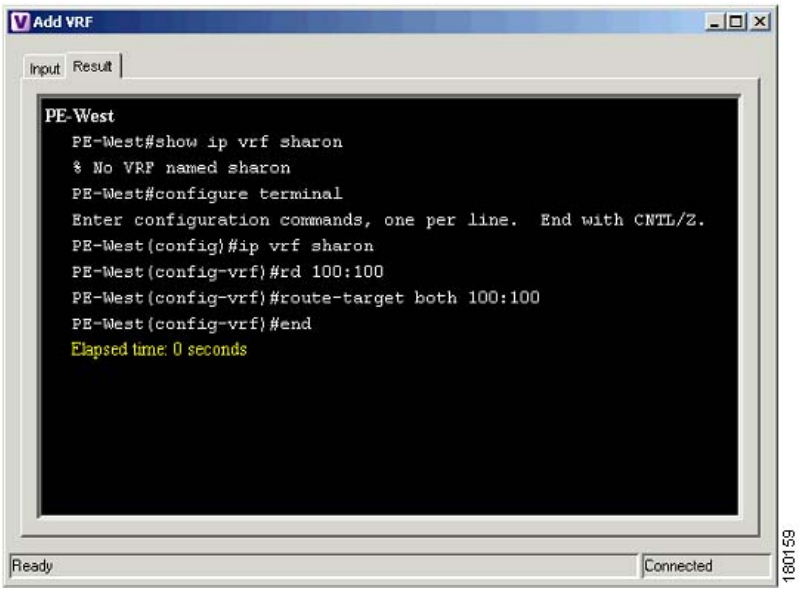
- Preview—Enables the user to see how the command, including, the variables look before it is executed. The result is displayed in the **Result** tab, an example of which is displayed below.

Figure 2-8 View Command Before Executing



- Execute—Enables the user to view the results of the actual command that is being executed. The result is displayed in the **Result** tab.

Figure 2-9 Actual Command Results



- Close—Returns to the opening window of the Command Builder wizard.

Step 3 Click **Preview** to see how the command looks before it is executed.

or

Click **Execute** to view the actual results of the command that is being executed.

Step 4 Close the input form and display the Command Builder wizard.

Publishing Commands

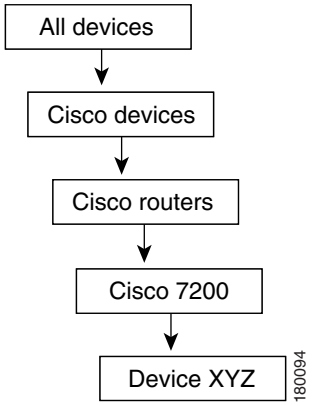
After the command has been defined and tested on a specific instance of a managed element it can be published and applied to wider scope of managed elements in the network.

The Command Builder Hierarchy Manager dialog box enables the user to publish the command to one or more locations across the inheritance hierarchy (as defined in the system). In other words the user defines the scope where the command will be applied in the hierarchy.

Different variations of a command can be used for different managed elements and network elements, where the implementation of the command is different for each managed element or network element.

An example of an inheritance hierarchy is displayed below. In this example, the top level of the hierarchy is All devices and the lowest level of the hierarchy is Device XYZ.

Figure 2-10 Inheritance Hierarchy Example



When a command is published to a node in the hierarchy, this overrides any inherited command from a higher level, and automatically applies to all its children. For example, if a command is published to Cisco 7200 it will override any variant of this command which is defined at a higher level, and will be assigned to all devices of type Cisco 7200 in the system.

To publish a command:

Step 1 Select the required command in the Command Builder wizard table.

Step 2 Click **Hierarchy Manager** on the toolbar of the Command Builder wizard.

or

Select **Hierarchy Manager** from the Tools menu or shortcut menu.

The Hierarchy Manager dialog box is generated and displayed.

Note If user-friendly VNE names exist in the schema then the hierarchy manager table will display these user-friendly registry location names in the VNE Hierarchy Location column. A user-friendlyVNE name is a hierarchy path that has been defined in the registry and is then displayed in the hierarchy manager table.

Each row that is displayed in the Hierarchy Manager dialog box represents a different level of the hierarchy. The rows are displayed in descending order; the top row is the highest level of the hierarchy and the bottom row is the lowest level of the hierarchy.

The following information is displayed in the table:

- Exist—When a node in the hierarchy is selected this indicates that a local variant of the command exists for that node.
- Registry Key—The hierarchy path, as defined in the registry.

The following tools are displayed in the Hierarchy Manager dialog box:

Button	Description
	Copies the command from a selected node in the hierarchy in order to copy it to another node in the hierarchy. A copy icon is displayed to the left of the selected node.
	Cuts the command from a selected node in the hierarchy in order to move it to another node in the hierarchy. A cut icon is displayed to the left of the selected node.
	Pastes the command that was copied or cut from a selected node in the hierarchy to another node in the hierarchy. A paste icon is displayed to the left of the selected node.
	Deletes the command from the selected node in the hierarchy. **Note** If the command has been deleted from all the nodes, the command will be removed from the list in the main dialog of the Command Builder Wizard.

The following button is displayed in the Hierarchy Manager dialog box:

- Close—Closes the Hierarchy Manager dialog box without publishing the command.

Step 3 Select the required node in the hierarchy from which you want to publish the command.

Step 4 Click **Copy** or **Cut** on the toolbar to copy or cut the command.

Step 5 Select the required node in the hierarchy where you want to publish the command.

Step 6 Click **Paste** on the toolbar to paste the command. The command is published to the selected node in the hierarchy.

Deleting a Command

Commands created by the user are by default always created as a local instance. A command that is defined locally is selected in the Command Builder wizard. The user can only delete commands that exist locally, namely, commands that have not yet been published.

To delete a command:

Step 1 Select the command that you want to delete in the table of the Command Builder wizard.

Step 2 Click **Delete Element** on the toolbar of the Command Builder wizard.

or

Select **Delete Element** from the File menu or shortcut menu.

The command is deleted and no longer displayed in the Command Builder table.

Importing and Exporting a Command

Command Builder enables the user to export (save) a full command definition to a file. The command definition can be imported (copied) later to another managed element.

In addition, the user can export and import a full command definition to a file and publish it to multiple places in the Hierarchy Manager window.

To export a command:

Step 1 Select the command that you want to export in the table of the Command Builder wizard.

Step 2 Click **Export Element** on the toolbar of the Command Builder wizard.

or

Select **Export Element** from the Tools menu or shortcut menu.

The Export dialog box is displayed.

Step 3 Select the version that you want to export in the table of the Export window, by checking the checkbox in the **Selected** column. The version is selected in the table.

Step 4 Click **OK**. The Export Property dialog box is displayed.

Step 5 Browse to the directory where you want to save the command.

Step 6 In the **File name** field, enter a name and extension (for example, **.txt**) for the command.

Step 7 Click **Save**. The command is saved in the selected directory. The Export dialog box is displayed.

Step 8 Click **Close**. The Command Builder wizard is displayed.

To import a command:

Step 1 Shortcut on the required managed element in the tree pane or context pane of the NetworkVision window.

or

Open the Inventory window for the required managed element and shortcut on the required network element.

Step 2 Select **Edit | Command Builder** from the shortcut menu. The Command Builder wizard is displayed.

Step 3 Click **Import Element** on the toolbar of the Command Builder wizard.

or

Select **Import Element** from the Tools menu or shortcut menu.

The Import Element dialog box is displayed.

Step 4 Browse to the directory and command that you want to import.

Step 5 Click **Open**. The Import elements window is displayed.

Step 6 Select the version that you want to import in the table of the Import Elements window. The version is selected in the table.

Step 7 Click **OK**. The Command Builder wizard is displayed.

Step 8 Click **Close**. The command is imported to the selected managed element or network element and displayed in the opening window of the Command Builder wizard.

Reviewing the Command History

Every command that is executed is logged in the Cisco ANA event database. The command's execution history can be viewed using the EventVision application. For more information about EventVision see the *Cisco Active Network Abstraction EventVision User Guide*.

Closing the Command Builder

When the user has finished working with the Command Builder the user can close the Command Builder wizard.

To close the Command Builder, click **Close**. The Command Builder wizard is closed.

CHAPTER 3

Creating an ANA Macro Language Command: An Example

This chapter describes creating an ANA Macro Language command *addvrf* command from start to finish.

Creating a Command

This section describes how to create an ANA Macro Language command from beginning to end. The example used here describes how to create the *addvrf* command.

To create the add vrf command:

Step 1 Shortcut on a managed element in the tree pane or context pane of the NetworkVision window.

or

Open the Inventory window for the required managed element and shortcut on the required network element.

Step 2 Select **Management | Command Builder** from the shortcut menu. The Command Builder wizard is displayed.

Step 3 Click **New** in the Command Builder wizard. The New Command dialog box is displayed.

Step 4 Define the command identification information, as follows:

- Name—addvrf
- Caption—Add VRF
- Menu Visible—Selected
- Menu Path—VRF Commands
- Context IMO—Automatically displayed
- Language—ANA Macro Language
- Protocol—By default Telnet is selected

Step 5 Click **Next**. The Command Authorizations dialog box is displayed.

Step 6 Select the security access role **Administrator**.

Step 7 Click **Next**. The User Input Arguments dialog box is displayed.

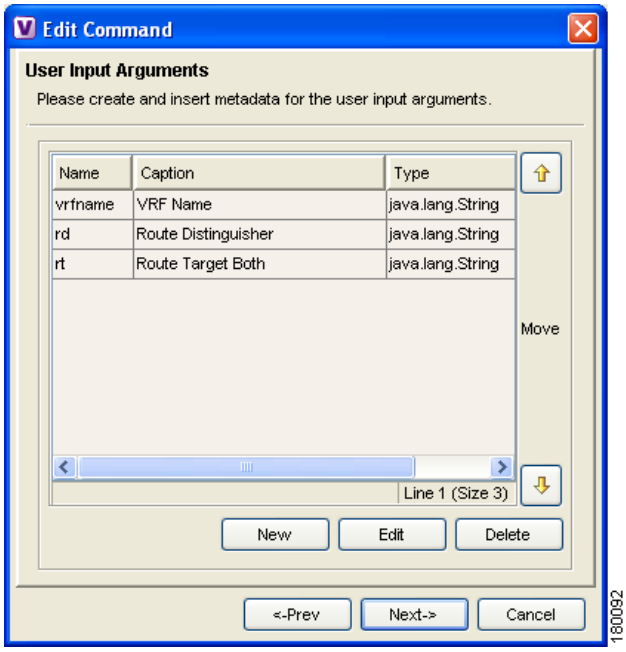
Step 8 Click **New**. The Add/Edit User Argument for *addvrf* Command dialog box is displayed.

Step 9 Enter the following information for the parameters:

	Parameter 1	Parameter 2	Parameter 3
Name	vrfname	rd	rt
Caption	VRF Name	Route Distinguisher	Route Target Both
Type	String	String	String
Width	15	15	15
Visible	Select	Select	Select

Step 10 Click **OK** in the Add/Edit User Argument dialog box after each parameter. The new user-defined input parameters are displayed in the User Input Arguments dialog box.

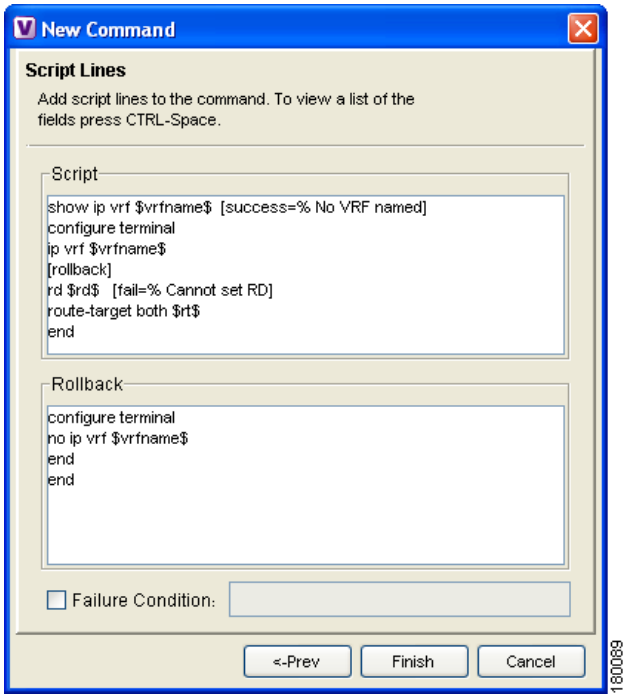
Figure 3-1 User Input Arguments Dialog Box



Step 11 Click **Next**. The Script Lines dialog box is displayed enabling the user to add the script lines.

Step 12 Add the script lines, as shown in the example below.

Figure 3-2 Script Lines Example



Step 13 Click **Finish**. The Create Command dialog box is displayed. LEDs indicate the progress or status of the command as it is being saved to the registry.

Step 14 Click **Close** when the command is complete. The newly created *addvrf* command is displayed in the Command Builder table.

Step 15 Select the *addvrf* command in the Command Builder table.

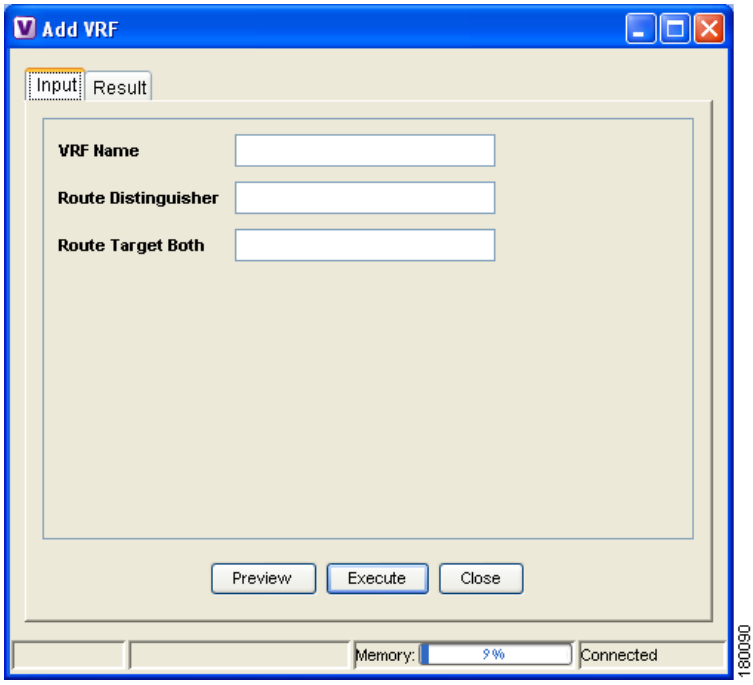
Step 16 Click **Run Command** on the toolbar of the Command Builder wizard.

or

Select **Run Command** from the Tools menu or shortcut menu.

The input form is generated and displayed.

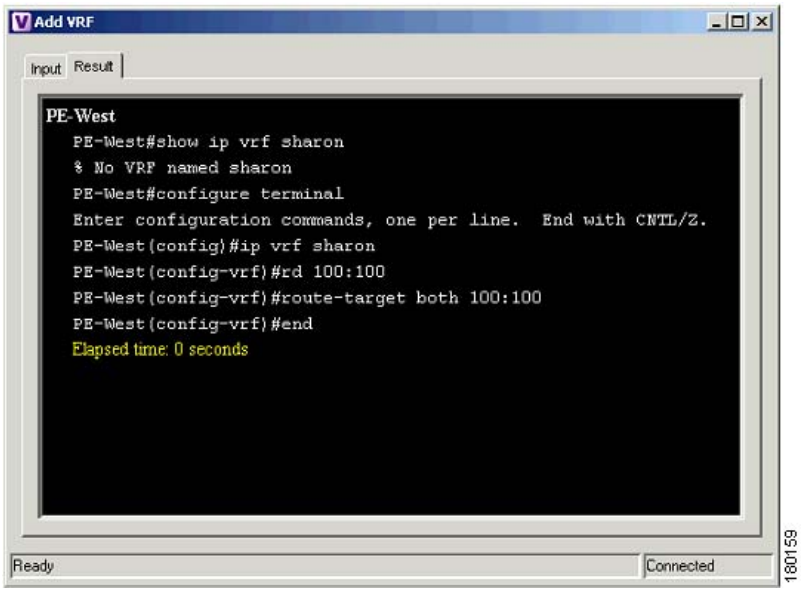
Figure 3-3 Input Form



Step 17 Enter values for the **VRF Name**, **Route Distinguisher**, and **Route Target Both** fields.

Step 18 Click **Execute** to view the actual results of the *addvrf* command that is being executed.

Figure 3-4 Results of the Command



Step 19 Close the input form and display the Command Builder wizard.

CHAPTER 4

Running Command Scripts

This chapter describes running command scripts.

Note Read the *Cisco Active Network Abstraction BQL User Guide* as a prerequisite to understanding this appendix.

For more information about ANA Macro Language scripts see Defining ANA Macro Language Script Lines, page 2-13.

Syntax

Item	Code and Explanation

General Syntax	`<command name="Command Script Name ">`
	`  <param name="oid">`
	`  <value> OID </value>`
	`  </param>`
	`  <param name="Parameter name 1">`
	`  <value>`
	`  Parameter Value`
	`  </value>`
	`  </param>`
	`  <param name="Parameter name 2">`
	`  <value>`
	`  Parameter Value`
	`  </value>`
	`  </param>`
	`</command>`

Command Script Name	Each command script is considered as BQL command. Thus, the name of the script is used as command name.
OID	OID format as explained in the *Cisco Active Network Abstraction BQL Guide*. The OID specifies the IMO object with which the command is associated.
Parameter name 1 (2) (3) …	The value of the input parmeters. A script can have none or many input parameters.

Note A command is always associated with an IMO object that enables it to use the properties of the object inside the script lines. For example, by selecting a Port object, the port's properties such as *portAlias*, and *status* are automatically made available to the script, but the command can access any other entity in the device.

Output

The output of a script is the IMO object: "IScriptResult". See IScriptResult, page 4-3 for a description of this object.

Examples

ChangePortStatus

The following example runs the script ChangePortStatus. This script changes the status of specified port. The parameter "portstatus" specifies the desired status of the port.

```
<command name="ChangePortStatus">
    <param name="oid">
<value>{[ManagedElement(Key=PE_South)][PhysicalRoot][Chassis][Slot(SlotNum=1)][Module][
Port(PortNumber=FastEthernet1/0)]}</value>
    </param>
    <param name="portstatus">
        <value>no shutdown</value>
    </param>
</command>
```

Add_VRF

The following example runs the script Add_VRF. This script adds new VRF to specified router. It receives the parameters: vfrName, rt & rd.

```
<command name="Add_VRF">
    <param name="oid">
        <value>{[ManagedElement(Key=P-North)]}</value>
    </param>
    <param name="rt">
        <value>333:333</value>
    </param>
    <param name="vrfName">
        <value>sharona2</value>
    </param>
    <param name="rd">
        <value>445:445</value>
    </param>
</command>
```

The script code in ANA Macro Language is as follows:

```
show ip vrf $vrfName$  [success= No VRF named $vrfName$]
[rollback]
[activity=Creating The VRF]
config terminal  [prompt=(config)]
ip vrf $vrfName$  [prompt=(config-vrf)]
[activity=Setting the route destinguisher]
rd $rd$  [fail=% Cannot set RD, check if it's unique]
route-target both $rt$
end
```

RunScript

The following example runs the script RunScript. This script enables the system to login to a **Cisco** device using the specified user and the script is then executed. It takes the parameters: a cid of a script, the username and password for the device.

```
<?xml version="1.0" encoding="UTF-8"?>
<command name="RunScript">
<param name="cid">
            <value>
<command name="ChangePortStatus">
    <param name="oid">

<value>{[ManagedElement(Key=CISCO-80.80.80.60)][PhysicalRoot][Chassis][Slot(SlotNum=0)]
[Module][Port(PortNumber=Ethernet0/1)]}</value>
    </param>
    <param name="portstatus">
        <value>no shutdown</value>
    </param>
</command>
</value>
</param>
<param name="userName">
            <value>tomer</value>
</param>
<param name="password">
            <value>tom1</value>
</param>
</command>
```

Script Results

IScriptResult

IScriptResult is the IMO object that holds the result of a script execution.

The object properties are:

- ExecutionTime—The time it took to run the script in milliseconds.

- StatusEnum—The status of the result represented in numeric values as follows:
  - 0 = Unknown
  - 1 = Success
  - 2 = Failure
  - 3 = Failure: device user switch unsupported
  - 4 = Failure: invalid credentials for device user switch
- FailedActivity—If the script failed this will hold one of the script labels defined in the script describing the part of the activity that failed.
- ExecutionSequence—Holds an array of IScriptEvent that holds the execution sequence that occurred in the script.

IScriptEvent

IScriptEven is the IMO object that describes a single event script that was run.

The object properties are:

- EventType—The type of the event. Represented in numeric values as follows:
  - 0 = Telnet Prompt
  - 1 = Telnet Command
  - 2 = Telnet Reply
  - 3 = SNMP Get Command
  - 4 = SNMP Set Command
  - 5 = SNMP Result
  - 6 = Exception
  - 7 = User Remark
- Message—The message content of the event.

Successful Example

The following example shows the returned result of successful execution of the Add_VRF script, see Add_VRF, page 4-2:

```
<IScriptResult>
  <ID type="Oid">{[ScriptResult(ScriptName=Add_VRF)(Sequence=1125562053773)]}</ID>
  <ExecutionSequence type="IMObjects_Array">
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=1)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North#show ip vrf sharona2</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=2)]}</ID>
      <EventTypeEnum type="Integer">2</EventTypeEnum>
      <Message type="String">% No VRF named sharona2
</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=3)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North#config terminal</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=4)]}</ID>
      <EventTypeEnum type="Integer">2</EventTypeEnum>
      <Message type="String">Enter configuration commands, one per line.  End with
CNTL/Z.</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=5)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North(config)#ip vrf sharona2</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=6)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North(config-vrf)#rd 445:445</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=7)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North(config-vrf)#route-target both 333:333</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=8)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North(config-vrf)#end</Message>
    </IScriptEvent>
  </ExecutionSequence>
  <ExecutionTime type="Long">852</ExecutionTime>
  <FailedActivity type="">Null</FailedActivity>
  <StatusEnum type="Integer">1</StatusEnum>
</IScriptResult>
```

Unsuccessful Example

Here is an example of the same script that failed to execute:

```
<IScriptResult>
  <ID type="Oid">{[ScriptResult(ScriptName=Add_VRF)(Sequence=1125562233194)]}</ID>
  <ExecutionSequence type="IMObjects_Array">
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=1)]}</ID>
      <EventTypeEnum type="Integer">1</EventTypeEnum>
      <Message type="String">P-North#show ip vrf sharona2</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=2)]}</ID>
      <EventTypeEnum type="Integer">2</EventTypeEnum>
      <Message type="String">  Name                              Default RD
Interfaces
  sharona2                          445:445
</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=3)]}</ID>
      <EventTypeEnum type="Integer">2</EventTypeEnum>
      <Message type="String">P-North#</Message>
    </IScriptEvent>
    <IScriptEvent>
      <ID type="Oid">{[ScriptEvent(Index=4)]}</ID>
      <EventTypeEnum type="Integer">6</EventTypeEnum>
      <Message type="String"> ^ Failed to find the text ' No VRF named sharona2' in the
device reply!, script terminated.</Message>
    </IScriptEvent>
  </ExecutionSequence>
  <ExecutionTime type="Long">162</ExecutionTime>
  <FailedActivity type="String" />
  <StatusEnum type="Integer">2</StatusEnum>
</IScriptResult>
```

APPENDIX A

ANA Macro Language

This appendix describes the ANA Macro Language, its syntax, how to use parameters, pragmas and a detailed example for writing ANA Macro Language scripts. This guide is intended for use by programmers who want to write command scripts that are executed within the Cisco ANA activation framework.

This appendix includes the following sections:

- What Are ANA Macro Language Scripts?, page A-1—Provides a definition of ANA Macro Language programming language and command scripts.
- Properties Available From the IMO Context, page A-2—Describes a script's Information Model context.
- Specifying and Using Parameters, page A-2—Describes the user-defined and built-in parameters used in script lines.
- Supported Pragmas, page A-4—Describes the inline ANA Macro Language directives supported by the language.
- Example, page A-9—Provides a full example of an ANA Macro Language based command script.

What Are ANA Macro Language Scripts?

ANA Macro Language scripts are a simple sequence of Telnet commands, runtime replaced input arguments and inline execution directives that are executed sequentially as telnet configuration commands on a networking device. ANA Macro Language script lines are evaluated in runtime for argument replacements that result in generation of a Telnet device configuration command that can be sent to the device. The reply of executing each command line is validated according to the inline directives that can abort and rollback the script or continues executing the next script line. ANA Macro Language scripts can be created using the Command Builder or can be provided externally using the Cisco ANA BQL API.

An ANA Macro Language script is usually made of a command script and a rollback script. The programmer can specify that if a command script has been detected to have failed, a rollback script would be called.

When defining ANA Macro Language scripts the programmer can:

- Import or paste scripts from external sources.
- Define inline directives (pragmas) for validating the network element's reply.
- Define a rollback script for undoing failed commands.

Properties Available From the IMO Context

The script IMO context makes the Cisco ANA Information Model Objects available to the script as built-in arguments. A script IMO context can be any object represent-able by the Cisco ANA IMO, ranging from a managed element, through port connector to a routing entry.

Example IMO contexts can include:

- Managed device
  - **IMO name**—IManagedElement
  - **Example properties**—CommunicationStateEnum, DeviceName, ElementType
- Port
  - **IMO name**—IPortConnector
  - **Example properties**—portalias, location, ifindex

For more information about ANA Macro Language Built-in parameters see Built-In Parameters, page A-4.

Specifying and Using Parameters

ANA Macro Language supports two types of script parameters. User-defined parameters and built-in parameters, both are replaced at runtime. In the Command Builder GUI all parameters (both built-in and user-defined ones) are available during command editing via a selection list.

Note To view all the user-defined and built-in parameters in the Command Builder application press **Ctrl + Spacebar** to open the selection list of available arguments (containing both the user-defined input argument and the built-in properties of the IMO context).

ANA Macro Language represents both types of parameters in script lines within dollar symbols, namely, **$...$**. For example, in a VRF configuration command, the input variable *vrfName* can be defined as *ip vrf $vrfName$*

Note Timeout for pragmas and scripts is supported using BQL. This adds a timeout type integer defined in milliseconds. It is recommended that if you change the timeout for the pragma you also change the timeout for the script.

Note An example of a timeout for a pragma is `route-target both $rt$ [timeout=2000]`

Note An example of a timeout for a script is `<Timeout type="Integer">5000</Timeout>`

User-Defined Parameters

User-defined input parameters must be defined up front. A parameter specification includes parameter name, type and even an optional default value. User-defined parameters can be defined using the Command Builder or through the Cisco ANA API.

The following is a complete list of the user-defined parameter properties:

Property	Explanation
Name	Parameter name. Can contain only letters, digits, "-" and "_". Unique within the script scope.
Caption	Parameter display name. Visible in the Command Builder script execution window.
Type	String, Integer, IPSubnet, Combo, IP, Float, Long.
Width	Field width, in characters. Relevant for the Command Builder script execution window.
Visible	Show this parameter in the window or not. Relevant for the Command Builder script execution window.
Default	A default value for the parameter. **Note** This property is only available through the Command Builder GUI.

Note Some parameter properties are relevant only for the script data entry window in the Command Builder.

During runtime the script is executed via a BQL command. As with all BQL commands if the argument types do not match, then an exception is returned to the user.

User defined parameters values can be provided in one of the following ways:

- Using flow-through activation the input parameters are provided as part of the API before they are sent to the Virtual Network Element (VNE).
- Run from NetworkVision as a GUI-based command. The programmer provides the input before they are sent to the VNE. For example, by typing a value or selecting one from the dropdown list.

Multiple Formats for IP Subnet Parameters

ANA Macro Language scripts support multiple formats for IP subnet parameters, as described in the table below, using the example 198.168.2.10 255.255.255.0:

#	Format	Description	Output
1	maskbits	The IP of the subnet converted to an integer value. Only bits.	30
2	ip	Only the IP without the mask.	198.168.2.10
3	mask	The IP of the subnet mask without the IP address.	255.255.255.0

#	Format	Description	Output
4	networkmask	The mask address converted to the network.	0.0.0.255
5	ipmaskbits	The IP and the value of the mask bits.	IP/30
6	ipmask	The IP mask. This is the default.	198.168.2.10 255.255.255.0
7	ipmasknot	The IP and the network address.	198.168.2.10 + 0.0.0.255

For example, routeadd$SB:IP$mask$SB:mask$. The same argument extracts the IP and then the subnet.

Built-In Parameters

Built-in parameters are the built-in properties available in IMO arguments of the IMO context (for example, *portalias*, *status* etc.), which are automatically set to their runtime value during execution. The built-in properties include IMO attributes, OID attributes, and instrumentation data.

For a complete list of the available built-in parameters related to the IMO context see the *IMO Reference Guide*.

Note To view all the user-defined and built-in parameters in the Command Builder application press **Ctrl + Spacebar** to open the selection list of available arguments (containing both the user-defined input argument and the built-in properties of the IMO context).

Supported Pragmas

The programmer can insert *inline directives (pragmas)* in the script lines for finer-granularity control. Pragmas are enclosed within square brackets, namely, **[...]**. The ANA Macro Language supported pragmas are listed below:

Pragma	Short Description	Refer to...
Success	Line-specific success check	Success
Fail	Line-specific failure check	Fail
Prompt	Line-specific prompt assertion validation	Prompt
Full prompt	Full prompt line-specific prompt assertion validation	Full Prompt
Rollback	Rollback enable or disable	Rollback
Activity	Script remarks. These would also help to determine the failure location	Activity
Enum	Defining enumerated value substitution	Enum

Note Wherever the carriage return character is required in the middle of a command line, the programmer should use the escape sequence **&cr**.

Note It is possible to use multiple pragmas in a single line, in which case all the pragmas will be analyzed. If the same type of pragma is repeated, only the last one will be used.

Success

Description

A *success* pragma is validated against the script line reply. The *success* pragma verifies that a required substring exists in the reply. If the substring is not found, the script fails.

Syntax

```
[success=<string>]
```

Where **<string>** represents the expected return value from the device, the **<string>** can be simple text or can contain arguments that will be replaced in runtime.

Directives

The pragma will be successful and the script will continue **only if** the **<string>** is found in the device reply.

The pragma will fail if the **<string>** does not exist in the reply.

The **<string>** can be a regular expression; it does not necessarily have to be an exact string to match.

Examples

The following example verifies that the specified VRF $newVrf$ does not already exist:

```
show ip vrf $newVrf$  [success=% No VRF $newVrf$]
```

Placing *Trial* as the *newVRF*, this pragma will yield success if the device reply would contain *% No VRF Trial*.

Fail

Description

A *fail* pragma is validated against the script line reply. The *fail* pragma verifies that a required substring does NOT exist in the reply.

Syntax

```
[fail=<string>]
```

Where **<string>** represents the value that should not be included in the device reply, the **<string>** can be simple text or can contain arguments that will be replaced in runtime.

Directives

The script will fail if **<string>** is found in the device reply.

The script will continue if the **<string>** does not exist in the reply.

The **<string>** can be a regular expression; it does not necessarily have to be an exact string to match.

Example

The following example sets a route distinguisher.

```
rd $newRD$   [fail=% Cannot set RD $newRD$]
```

Placing **60:60** as the newRD, this pragma will yield failure only if the device reply would contain **=% Cannot set RD 60:60**.

Prompt

Description

A *prompt* pragma is validated against the next telnet command *prompt*. The prompt pragma verifies that the suffix of the prompt equals the given string. If the suffix differs from the string, the script fails.

Syntax

```
[prompt=<prompt>]
```

Where **<prompt>** represents the new expected prompt, the **<prompt>** can be simple text or can contain arguments that will be replaced in runtime before sending to the device.

Directives

The pragma will be successful and script execution continues only if the **<prompt>** is found as the suffix of the device prompt.

The pragma will fail if the **<prompt>** is not found in the suffix of the device prompt.

Example

The following example changes the telnet prompt and validates the change in the newly returned telnet prompt.

```
configure terminal  [prompt=(config)]
```

This pragma will yield success only if the next device prompt would end with **(config)#**.

Full Prompt

Description

A *full prompt* pragma is validated against the next telnet command prompt. The *full prompt* pragma verifies that the prompt equals the given string. If the prompt differs from the string, the script fails.

Syntax

```
[prompt=^<prompt>]
```

Where **<prompt>** represents the expected full prompt, the **<prompt>** can be simple text or can contain arguments that will be replaced in runtime before sending to the device.

Directives

The pragma will be successful and script execution continues only if the next full prompt equals **<prompt>**.

The pragma will fail if the next prompt does not equal **<prompt>**.

Example

The following example changes the telnet prompt and validates the change in the newly returned telnet prompt.

```
configure terminal  [prompt=^router(config)#]
```

This pragma will yield success only if the next device prompt would match **router(config)#** exactly.

Rollback

Description

Rollback scope—the pragma **[rollback]** determines that rollback will be executed only upon failures from this point onwards.

Note Be sure the rollback script restores the devices prompt to its original value before the script was initiated.

Directives

If the script fails after the **[rollback]** marker then rollback is executed.

Note If the rollback script fails, no additional actions can be performed.

Activity

Description

Set the text that, if the script fails, will appear in the script's result as the name of the activity that failed. The failed activity name (label) appears in the returned result and also in the provisioning event that is generated.

Syntax

```
[activity=<activity>]
```

Where **<activity>** stands as an inline remark comment, the **<activity>** can be simple text or can contain arguments that will be replaced in runtime before sending to the device.

Directives

When a failure occurs later in the script, the user is notified of the error according to the "activity name."

Example

```
[activity=now adding the vrf]
```

Enum

Description

Defines the values that are used when substituting parameter names into a Telnet string.

Directives

The pragma will be successful only if the user inputs one of the values in the list.

The pragma will fail if the user does not input one of the values in the list.

Example

The enum pragma appears at the top of the script:

```
[enum RouteTargetTypeEnum 0=export;1=import]
```

Later in the script, the parameter RouteTargetTypeEnum is used:

```
no route-target $RouteTargetTypeEnum$ $RouteTarget$
```

The value that is substituted into the Telnet command for **$RouteTargetTypeEnum$** is "export" or "import" instead of "0" or "1".

Example

The following script and rollback script performs an *Add VRF* configuration. The script uses user-defined arguments to represent the VRF name, route target and route distinguisher, several types of pragmas to validate the device reply, remarks in the command script and rollback script.

Command Script

```
[enum rd 1=60:60;2=80:80]
show ip vrf $vrfName$ [success=% No VRF named $vrfName$]
[activity=prepare for VRF creation]
config terminal [success=Enter configuration commands, one per line.  End with CNTL/Z.]
[prompt=(config)]
ip vrf $vrfName$ [prompt=(config-vrf)]
[rollback]
[activity=create VRF]
rd $rd$ [fail=% Cannot set RD, check if it's unique]
route-target both $rt$
end
```

Rollback Script

```
config terminal
no ip vrf $vrfName$
end
```

The following table lists the user-defined argument definitions used in the script:

Name	Type	Default	Explanation	Example
vrfName	String	N/A	The VRF name. The value provided for this argument will be used as the VRF table name.	Manhattan
rt	String	N/A	The VRF route target, in the format of *integer:integer*. The value provided for this argument will be used as is for the device configuration.	60:60
rd	String	1	In this example, the system administrator would like the route distinguisher to be based on the pre-defined enumerated values list. Hence, the route distinguisher will be provided in the format of an integer to be used as a lookup table key, and not x:y.	1, 2 or any valid value according to the enum pragma

The following table provides an explanation of the command script line-by-line:

Script Line	Explanation
[enum rd 1=60:60;2=80:80]	The line enumerates the possible values of the route distinguisher argument.
show ip vrf $vrfName$ [success=% No VRF named $vrfName$]	Verify if the requested VRF already exists. Continue to create the VRF only if the requested VRF name is not found.
[activity=prepare for VRF creation]	Remark to state that the following section is preparation for VRF creation.
config terminal [success=Enter configuration commands, one per line. End with CNTL/Z.] [prompt=(config)]	Change mode command. Continue to the next command if the *success* pragma string is found in the device reply and the *prompt* changes to *config*.
ip vrf $vrfName$ [prompt=(config-vrf)]	Change mode command. Continue to the next command if the *prompt* changes to *config-vrf*.
[rollback]	Placeholder to state that rollback should be executed only if a subsequent script line fails.
[activity=create VRF]	Remark to state that the following section is actually the VRF creation.
rd rd [fail=% Cannot set RD, check if it's unique]	Set the route distinguisher. If this command will fail the rollback script will be called.
route-target both rt	Set the route target. If this command will fail the rollback script will be called.
end	Change mode command. Return to normal (“enable”) mode.

The following table provides an explanation of the activation rollback script line-by-line:

Script Line	Explanation
config terminal	Set the device to *terminal* mode.
no ip vrf $vrfName$	Delete the VRF from the device.
end	Change mode command. Return to normal (“enable”) mode.

Running the Script

The script is executed with the following input arguments:

```
vrfName=Trial
rd=2
rt=60:60
```

The telnet commands as would be sent to the device (preview):

```
show ip vrf Trial
config terminal
ip vrf Trial
rd 80:80
route-target both 60:60
end
  ------Rollback------
config terminal
no ip vrf Trial
end
```

Full session:

```
vrfName=Trial
rd=2
rt=60:60

PE-North#show ip vrf Trial
% No VRF named Trial
PE-North#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-North(config)#ip vrf Trial
PE-North(config-vrf)#rd 80:80
PE-North(config-vrf)#route-target both 60:60
PE-North(config-vrf)#end
```

Running the script with the same input values again (VRF already exists, the command stops after VRF name verification)

```
PE-North#show ip vrf Trial
  Name                              Default RD          Interfaces
  Trial                             80:80
PE-North#
 ^ Failed to find the text '% No VRF named Trial' in the device reply!, script terminated.
```

Running the script with a different VRF name (same RT & RD) (VRF would begin to be created and then rollback due to RD already in use)

```
vrfName=Trial2
rd=2
rt=50:50

PE-North#show ip vrf Trial2
% No VRF named Trial2
PE-North#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-North(config)#ip vrf Trial2
PE-North(config-vrf)#rd 80:80
% Cannot set RD, check if it's unique
PE-North(config-vrf)#
 ^ Error in activity 'create VRF'.
 ^ Found the text '% Cannot set RD, check if it's unique' in the device reply!, script
terminated.
-----Invoking Rollback-----
PE-North#config terminal
Enter configuration commands, one per line.  End with CNTL/Z.
PE-North(config)#no ip vrf Trial2
% IP addresses from all interfaces in VRF Trial2 have been removed
PE-North(config)#end
```

APPENDIX B

Bean Shell Commands

This appendix describes the methods that should be used for Bean Shell in Cisco ANA commands when you want to interact with devices. In addition, it provides Telnet and SNMP environment object examples.

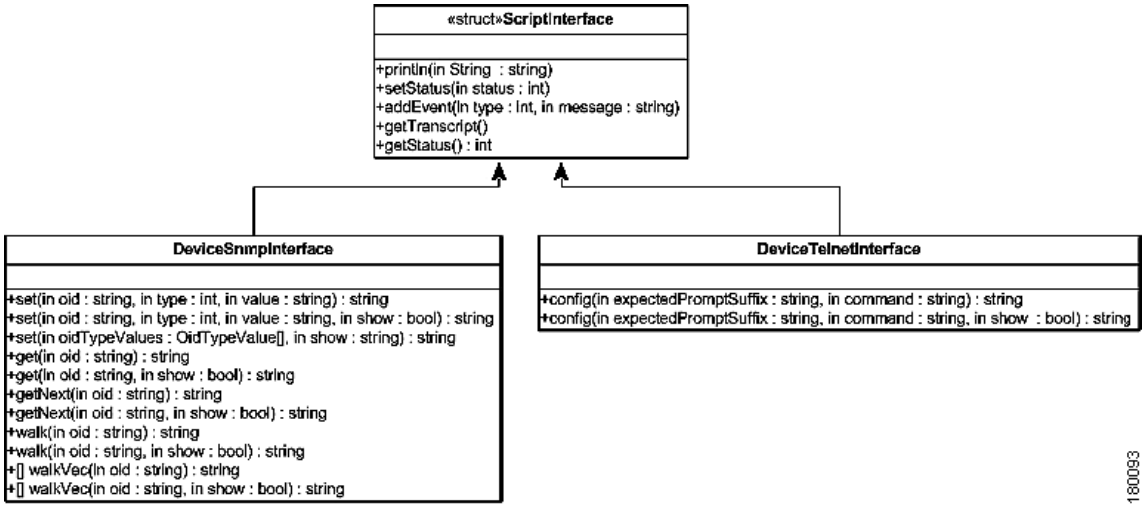
Note Important: Unlike ANA Macro Language, in Bean Shell user arguments inventory properties should NOT be embedded within $...$ symbols.

This appendix includes the following sections:

- Telnet Examples, page B-2—Provides examples of the available predefined Telnet environment objects.
- SNMP Examples, page B-2—Provides examples of the available predefined SNMP environment objects.

The diagram below describes the methods that should be used for Bean Shell in Cisco ANA commands when interacting with devices for Telnet and SNMP interfaces.

Figure B-1 Bean Shell Methods



Note For setStatus 1=success and 2=failure.

Telnet Examples

This section provides examples of the available predefined Telnet environment objects that you can use to interact with a device.

- telnetInterface.config (“prompt”,”telnet command”)—Where “prompt” is the expected prompt after command completes and “telnet command” is the actual command to be execute
- telnetInterface.setStatus (*1 or 2*)—Where 1=success and 2=fail, used for example to signal to work flow manager that a configuration command has succeeded or failed

SNMP Examples

This section provides examples of the available predefined SNMP environment objects that you can use to interact with a device.

snmpInterface.get (“OID”,true/false);

Gets OID, and displays or hides results

snmpInterface.getNext (“OID”,true/false);

Gets next OID, displays or hides results

snmpInterface.set (“OID”,variable type,”value”);

Sets OID to specified value

snmpInterface.walk(“OID”,true/false);

Returns vector of strings

snmpInterface.setStatus (1 or 2);

Where 1=success and 2=fail, used for example to signal to work flow manager that a configuration command has succeeded or failed

For additional information about general scripting language refer to http://www.beanshell.org/.