Command Manager and Command Builder: Macro Language and Beanshell Reference

Prime Network Command Manager and Command Builder support two languages for writing command scripts:

- Prime Network Macro Language—Provides simple sequences of Telnet commands, runtime-replaced user-defined input parameters, and inline execution directives that are executed sequentially as Telnet configuration commands on an NE. See Prime Network Macro Language, page C-1.
- BeanShell—Provides a fully programmatic logic via scripting language (including conditions, loops, and external files). See BeanShell Commands, page C-11.

Prime Network Macro Language

These topics describe the Prime Network Macro Language and its syntax, how to use parameters and pragmas, and a detailed example for writing Prime Network Macro Language scripts.

Topics include:

- What Are Prime Network Macro Language Scripts?, page C-1
- Properties Available from the IMO Context, page C-2
- Specifying and Using Parameters, page C-2
- Supported Pragmas, page C-4
- Example, page C-8

What Are Prime Network Macro Language Scripts?

A Prime Network Macro Language script is 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. Prime Network Macro Language script lines are evaluated in runtime

for argument replacements that result in the generation of a Telnet device configuration command that can be sent to the device. Each command line is validated according to the inline directives that can abort and roll back the script or continue executing the next script line. Prime Network Macro Language scripts can be created using Command Manager or Command Builder, or can be provided externally using the Prime Network BQL API.

A Prime Network Macro Language script is usually made of a command script and a rollback script. You can specify that if a command script fails, a rollback script is called.

When defining Prime Network Macro Language scripts, you 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 Prime Network Information Model Objects available to the script as built-in arguments. A script IMO context can be any object that can be represented by a Prime Network IMO, ranging from a managed element to a port connector to a routing entry. Example IMO contexts can include:

NE Type	IMO Name	Example Properties
Managed device	IManagedElement	CommunicationStateEnum, DeviceName, ElementType
Port	IPortConnector	portalias, location, ifindex

For more information about Prime Network Macro Language Built-in parameters, see Built-In Parameters, page C-4.

Specifying and Using Parameters

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

Note To view all user-defined and built-in parameters in the Command Manager and Command Builder applications, 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).

Prime Network Macro Language represents both types of parameters in script lines within dollar signs; for example, $...$. For instance, in a VRF configuration command, the input variable vrfName can be defined as ip vrf $vrfName$.

Note

- Timeouts for pragmas and scripts are supported using BQL. This adds a timeout type integer defined in milliseconds. We recommend that if you change the timeout for the pragma, you also change the timeout for the script.
- An example of a timeout for a pragma is `route-target both $rt$ [timeout=2000]`.
- 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 Command Manager or Command Builder, or through the Prime Network API.

Table C-1 provides a complete list of user-defined parameter properties.

Table C-1 Available User-Defined Parameters

Property	Explanation
Name	Parameter name. Can contain only letters, digits, hyphen (-), and underscore (_), and must be unique.
Caption	Parameter display name. Visible in the Command Manager and Command Builder script execution window.
Type	String, Integer, IPSubnet, Combo, IP, Float, Long.
Width	Field width, in characters. Relevant for the Command Manager and Command Builder script execution window.
Visible	Indicates whether or not the parameter appears in the window. Relevant for the Command Manager and Command Builder script execution window.
Tooltip	Tooltip for the command parameter. **Note** This property is only available through the Command Manager and Command Builder GUIs.
Default	A default value for the parameter. **Note** This property is only available through the Command Manager and Command Builder GUIs.
Required	Indicates whether the argument is mandatory or optional. **Note** This property is only available through the Command Manager and Command Builder GUIs.

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

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

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

- Using flow-through activation—The input parameters are provided as part of the API before they are sent to the VNE.
- Run from Prime Network Vision as a GUI-based command—You provide the input parameters before they are sent to the VNE; for example, by entering a value or choosing one from a drop-down list.

Multiple Formats for IP Subnet Parameters

Prime Network Macro Language scripts support multiple formats for IP subnet parameters, as described in Table C-2, using the example 198.168.2.10 255.255.255.0.

Table C-2 Formats for IP Subnet Parameters

#	Format	Description	Output
1	maskbits	The IP of the subnet converted to an integer value. Bits only.	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
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$SB:mask$ 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 (such as portalias or status), which are automatically set to their runtime value during execution. The built-in properties include IMO attributes, OID attributes, and instrumentation data.

Note To view all user-defined and built-in parameters in the Command Manager and 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

You can insert inline directives (pragmas) in the script lines for increased granularity control. Pragmas are enclosed within square brackets ([…]). Table C-3 lists the pragmas that Prime Network Macro Language scripts support.

Table C-3 Supported Pragmas

Pragma	Short Description	Refer to…
Success	Line-specific success check.	Success
Fail	Line-specific failure check.	Fail
Prompt	Line-specific prompt assertion validation.	Prompt
Rollback	Rollback enable or disable.	Rollback
Activity	Script remarks. These also help 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, use the escape sequence **&cr**.

Note You can use multiple pragmas in a single line; when this occurs, all pragmas are analyzed. If the same type of pragma is repeated, only the last one is 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. *<string>* can be simple text or can contain arguments that are replaced in runtime.

Directives

The pragma succeeds and the script continues only if *<string>* is found in the device reply.

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

<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 $newVrf$]
```

Using Trial for newVrf, this pragma succeeds if the device reply contains `% 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. `<string>` can be simple text or can contain arguments that are replaced in runtime.

Directives

The script fails if `<string>` is found in the device reply. The script continues if `<string>` does not exist in the reply.

`<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 $newRD$]
```

Using 60:60 for newRD, this pragma yields failure only if the device reply contains `=% Cannot set RD 60:60`.

Prompt

Description

A *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 new expected prompt. `<prompt>` can be simple text or can contain arguments that are replaced in runtime before being sent to the device.

Directives

The pragma is successful and script execution continues only if the next full prompt equals <prompt>. The pragma fails 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 yields success only if the next device prompt matches router(config)# exactly.

Partial Prompt

Description

A partial *prompt* pragma is validated against the next Telnet command prompt. The partial 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 expected full prompt. `<prompt>` can be simple text or can contain arguments that are replaced in runtime before being sent to the device.

Directives

The pragma is successful and script execution continues only if `<prompt>` is found as the suffix of the device prompt. The pragma fails if `<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 succeeds only if the next device prompt ends with `(config)#`.

Rollback

Description

A *rollback* pragma determines that rollback will be executed only upon failures from this point onward.

Note Be sure the rollback script restores the device 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

An *activity* pragma sets the text that, if the script fails, appears in the script's result as the name of the activity that failed. The failed activity name (label) appears in the returned result and in the provisioning event that is generated.

Syntax

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

where *<activity>* represents an inline remark comment. *<activity>* can be simple text or can contain arguments that are replaced in runtime before being sent to the device.

Directives

When a failure occurs later in the script, you are notified of the error by activity name.

Example

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

Enum

Description

An enum pragma defines the values that are used when substituting parameter names into a Telnet string.

Directives

The pragma is successful only if you input one of the values in the list. The pragma fails if you do 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 command script and rollback script perform an *Add VRF* configuration. The scripts use user-defined arguments to represent the VRF name, route target, and route distinguisher; several types of pragmas to validate the device reply; and 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
```

Table C-4 lists the user-defined argument definitions used in the script.

Table C-4 User-Defined Argument Definitions

Name	Type	Default	Explanation	Example
vrfName	String	N/A	The VRF name. The value provided for this argument is used as the VRF table name.	Manhattan
rt	String	N/A	The VRF route target, in the format *integer:integer*. The value provided for this argument is 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 predefined enumerated values list. Therefore, the route distinguisher is 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

Table C-5 provides an explanation of the command script line by line.

Table C-5 Command Script Explanation

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.

Table C-5 Command Script Explanation (continued)

Script Line	Explanation
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 prompt changes to config.
ip vrf $vrfName$ [prompt=(config-vrf)]	Change mode command. Continue to the next command if 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 fails, the rollback script is called.
route-target both rt	Set the route target. If this command fails, the rollback script is called.
end	Change mode command. Return to normal (enable) mode.

Table C-6 provides an explanation of the activation rollback script line by line.

Table C-6 Rollback Script Explanation

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 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
```

Rerunning the script with the same input values (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 but the same route target (RT) and route distinguisher (RD) (VRF creation begins and then is rolled back 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
```

BeanShell Commands

These topics describe the methods that should be used for BeanShell in Prime Network commands when you want to interact with devices:

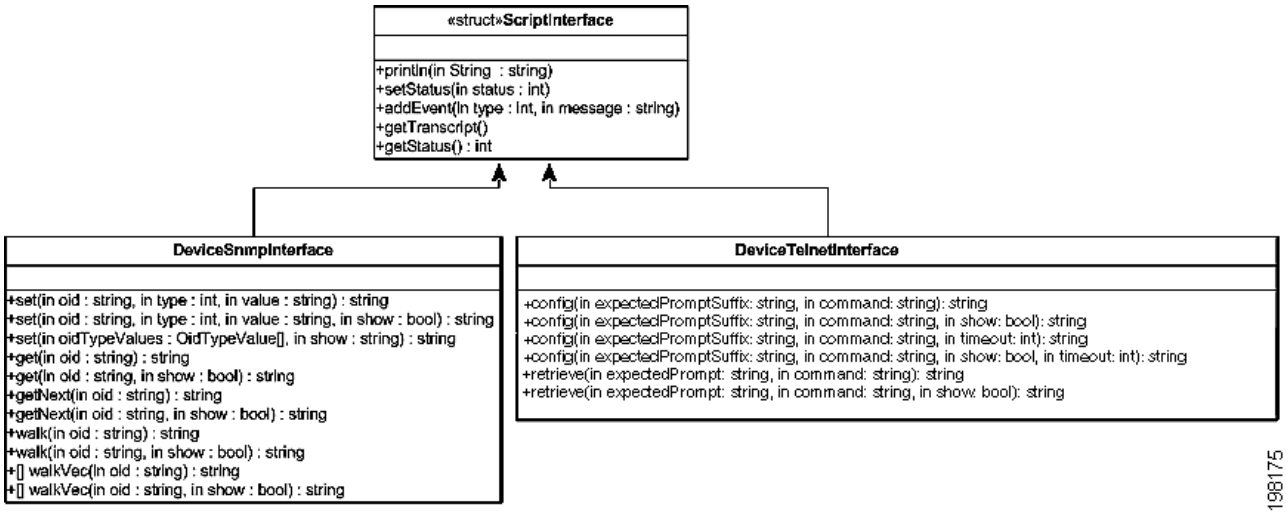
- Telnet BeanShell Commands, page C-12
- SNMP BeanShell Commands, page C-13

. In addition, it provides Telnet and SNMP environment object examples.

Caution Unlike Prime Network Macro Language, in BeanShell user arguments, inventory properties should *not* be embedded within dollar signs ($...$).

Figure C-1 presents the methods that should be used for BeanShell in Prime Network commands when interacting with devices for Telnet and SNMP interfaces.

Figure C-1 BeanShell Methods



Note For setStatus, 1 = success and 2 = failure.

Telnet BeanShell Commands

The following are examples of the available predefined Telnet environment objects that you can use to interact with a device:

- telnetInterface.config (*prompt*,*telnet_command*, true/false, *timeout*)—Where:
  - *prompt* is the expected prompt after command is executed.
  - *telnet_command* is the actual command to be executed.
  - true displays the results and false hides the results.
  - *timeout* is the CLI time out in milliseconds. The default value is 20000 milliseconds (20 seconds).
- telnetInterface.setStatus (1 or 2)—Where 1 = success and 2 = fail.
- telnetInterface.println—Used to print the output string on screen.

A timeout error reports the failure in the following format:

```
Unexpected error occurred during script execution:
receiveUntil(): general timeout expired(value=<elapsed_time_in_milliseconds>)
(<command_run>)
Elapsed time: <elapsed_time_in_seconds> seconds
```

where:

- *elapsed_time_in_milliseconds* is the length of the timeout in milliseconds.

- *command_run* is the command that was being executed when the timeout occurred.
- *elapsed_time_in_seconds* is the length of the timeout in seconds.

For example, a timeout error might read as follows:

```
Unexpected error occurred during script execution:
receiveUntil(): general timeout expired(value=10000)(copy tftp://171.69.75.3/radA020C.tmp
null:
Accessing tftp://171.69.75.3/radA020C.tmp...)
Elapsed time: 10 seconds
```

Reload Router Command Example

The following is an example of using BeanShell script to reload a router.

```
try {
  telnetInterface.config("[confirm]", "reload", false);
  telnetInterface.config("#", "\n", false, 2000);
}
catch (Exception e) {
  telnetInterface.println("Router will reload");
}
```

SONET Show Controller Data Command Example

The following is an example of the BeanShell implementation of the SONET Controller Data command that displays SONET controller data:

```
try {
    String sep = File.separator;
    source("." + sep + "scripts" + sep + "configuration" + sep + "cisco" + sep +
     "CiscoUtil.bsh");

    String strOid = oid.toString();
    String SEPARATOR = "PortNumber=POS";

    int startIdx = strOid.indexOf(SEPARATOR);
    startIdx = startIdx + SEPARATOR.length();
    int endIdx = strOid.indexOf(')', startIdx);

    String interfaceName = strOid.substring(startIdx, endIdx);

    telnetInterface.println("Running command: show controller sonet "+interfaceName);
    String res = telnetInterface.retrieve("#", "show controller sonet "+interfaceName);

    telnetInterface.println(res);

    telnetInterface.setStatus(XProvisioningConfigDeviceStatusMsg.STATUS_SUCCESS);

}   catch (Exception e) {
    telnetInterface.setStatus(XProvisioningConfigDeviceStatusMsg.STATUS_FAILURE);
    telnetInterface.println("Exception occurred during execution of the script " +
    e.getMessage() );
}
```

SNMP BeanShell Commands

The following are 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, and 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 = failure.

For additional information about general scripting language, see http://www.beanshell.org/.

TLI BeanShell Commands

The following is an example of the available predefined TL1 environment object that you can use to interact with a device:

- String config(String command, boolean show) throws Exception;
- String config(String command) throws Exception;
- String retrieve(String command) throws Exception;
- String retrieve(String command, boolean show) throws Exception;
- String waitForEvent(boolean show) throws Exception;
- String waitForEvent() throws Exception;
- String waitForEvent(int timeout) throws Exception;
- String waitForEvent(int timeout, boolean show) throws Exception;

For more information about TL1 commands, see

http://www.cisco.com/en/US/docs/optical/15000r9_1/tl1/sdh/beginners/guide/91e_tlbgn.html#wp44529

TL1 Command to Retrieve General NE Attributes Example

```
import com.sheer.metrocentral.framework.configuration.ScriptInterface;
import
com.sheer.metrocentral.framework.provisioning.messages.XProvisioningConfigDeviceStatusMsg;
import com.sheer.metrocentral.framework.configuration.ScriptInterface;
import
com.sheer.metrocentral.framework.provisioning.messages.XProvisioningConfigDeviceStatusMsg;
import java.util.HashMap;
import com.sheer.system.os.interfaces.Logger;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import com.sheer.util.*;
import java.lang.String;
ScriptInterface protocolInterface = deviceInterface;
String cmd="RTRV-NE-GEN:::123;";
String  result = protocolInterface.send(cmd,true);
```

```
protocolInterface.println(result);
```