CHAPTER 23

IMGW Device Module Development Toolkit

The Intelligent Modular Gateway (IMGW) device module development toolkit clearly defines the southbound interface of IMGW and provides a registration utility to allow you to register plug-in device modules into IMGW after the device module is installed onto the Cisco Configuration Engine.

This chapter analyzes the requirements of the IMGW device module development toolkit and describes the functionality that is offered by this toolkit.

Note You can also implement the device module in either shell scripts or Linux/Solaris executables as long as the device module conforms to IMGW southbound interface.

User Types

This toolkit is oriented to three types of users:

- *Plug-in Developer*—responsible for developing the device module that complies with the IMGW southbound interface defined in this toolkit
- *System Administrator*—responsible for the following:
  - Plug the device module into and out of the Cisco Configuration Engine
  - Register and de-register the plug-in device module
  - Update the device module on the Cisco Configuration Engine
- *Network Operator*—configures the device through the plug-in device module

Toolkit Usage

There are three common usages of this toolkit:

- Plug a device module into Cisco Configuration Engine and configure devices using the device module.
- Update a device module on the Cisco Configuration Engine and configure devices through the modified device module.
- Unplug a device module from the Cisco Configuration Engine.

Plug Device Module Into Cisco Configuration Engine

Step 1 The *Plug-in Developer* develops a device module conforming to the IMGW southbound interface defined in this toolkit to handle the given device type.

For information about the device module syntax, see "IMGW Southbound Interface" section on page 23-2.

Step 2 The *System Administrator* installs the device module onto Cisco Configuration Engine.

Step 3 The *System Administrator* runs the registration utility to register the device module into IMGW.

Step 4 The *Network Operator* configures devices through the device module.

Update Device Module on Cisco Configuration Engine

Step 1 The *Plug-in Developer* provides a new version of the device module.

Step 2 The *System Administrator* runs the registration utility to de-register the device module from IMGW.

If the device module you want to update is not registered, skip this step

Step 3 The *System Administrator* updates the device module with the new version on Cisco Configuration Engine.

Step 4 The *System Administrator* runs registration utility to register the updated device module into IMGW.

Step 5 The *Network Operator* configures devices through modified device module.

Unplug Device Module from Cisco Configuration Engine

Step 1 The *System Administrator* runs the registration utility to de-register the plug-in device module from IMGW.

Step 2 The *System Administrator* uninstalls the plug-in device module from Cisco Configuration Engine.

IMGW Southbound Interface

When a command execution or a configuration update event is received by IMGW runtime, it will first retrieve device type information from the device information database. If the device module corresponding to device type and operation type (**CONFIG_UPLOAD** or **CONFIG_DOWNLOAD**) is registered, IMGW runtime forks a process to execute the proper plug-in program and pass the parameter list to the plug-in program.

The initial mapping information from the <*device type, operation type*> pair to the plug-in program is read from a configuration file into memory upon start up. When IMGW is running, the system administrator can still add, remove, or update the entries of mapping information by way of the toolkit registration utility.

The *System Administrator* can modify only the entries for non-legacy device modules. This restriction is enforced by IMGW runtime.

User Designed Device Module Specifications

A user-defined device module must conform to the IMGW southbound interface as specified in this section.

Config Event

<***plug-in program***> <*temp_logfile_name*> <*logging_level*> <*device_id*> <*action_type*> <*warning_logfile_name*> <*error_logfile_name*> <*hop_information_string*> <*configuration_file_name*> <*persistence*> <*operation_timeout_value*> <*prompt_timeout_value*>.

Exec Event

<***plug-in program***> <*temp_logfile_name*> <*logging_level*> <*device_id*> <*action_type*> <*hop_information_string*> <*command_to_be_executed*> <*command_arguments*> <*exec_response_logfile_name*> <*operation_timeout_value*> <*prompt_timeout_value*>.

Hop Test

<***plug-in program***> <*temp_logfile_name*> <*logging_level*> <*device_id*> <*action_type*> <*hop_information_string*> <*operation_timeout_value*> <*prompt_timeout_value*>.

Note All files specified for the IMGW southbound interface are managed by IMGW runtime and their file names are absolute path names.

Parameter Descriptions

Plug-in Program: The plug-in program that is executed in the child process forked by IMGW runtime. The system administrator gives this information to IMGW runtime during registration.

temp_logfile_name: The full path to the device module temporary log file, which should be used by the device module to log the processing history of one instance of operation (configuration download, command execution or hop test). This file is by default located at */tmp* directory on the Cisco Configuration Engine. After the plug-in program exits, IMGW runtime puts the content of this file into a centralized log file named */opt/CSCOimgw/bin/IMGW-DEVMOD_LOG* for debugging purpose, then unlinks this file.

logging_level: It could be verbose, error, or silent. This flag can be set up by running setup command on the host system. It is recommended that the device module log information into the file <*temp_logfile_name*> based on the specified logging level.

device_id: The identification of the device that is processed by the device module. It is passed in by the *cisco.mgmt.cns.config.load* or *cisco.mgmt.cns.exec.cmd* event.

action_type: It could be **config**, **exec**, or **hoptest**. Action type **config** notifies the device module to update the device configuration. Action type **exec** notifies the device module to execute a command on the device. Action type **hoptest** notifies the device module to test if the device is reachable by way of the hop information provided in <*hop_information_string*>. The device module should do the proper operation in response to this flag.

warning_logfile_name: The full path to the file that is used by the device module to log all warning messages and its corresponding configuration commands line numbers. This parameter is supplied by IMGW runtime only when the action type is **config** because the information in this file is only used to generate the response message to the *cisco.mgmt.cns.config.load* event if the configure succeeds with warnings. In order for the IMGW runtime to generate the proper response message, each warning message should begin a new line and be prefixed with the string of **LINE <***line number of the configuration command that causes the warning message***>:** An example of the warning file is as follows:

```
LINE 3: The interface has already been removed
.
.
.
LINE 7: The interface already exists.
```

The location of this file is under */tmp* on the host system. After the plug-in program exits, IMGW runtime puts the content of this file into the response event payload, then immediately unlinks this file.

error_logfile_name: The full path to the file that is used by the device module to log the occurrences of the error messages and their corresponding configuration command line numbers. This parameter is supplied by IMGW runtime only when the action type is **config** because the information in this file is only used to generate the response message to the *cisco.mgmt.cns.config.load* event if the configure fails. In order for the IMGW runtime to generate the proper response message, each error message should begin a new line and be prefixed with the string of **LINE <***line number of the configuration command that causes the error message***>**.

An example of the error file is as follows:

```
LINE 3: % Invalid input detected at
LINE 7: % Incomplete command
.
.
.
LINE 12: % The interface already exists
```

The location of this file is under */tmp* on the host system. After the plug-in program exits, IMGW runtime puts the content of this file into the response event payload, then immediately unlinks this file.

exec_response_logfile_name: The full path to the file that is used to log the output of command execution on the device. It is supplied by IMGW runtime only when the action type is **exec** and its location is under */tmp* on the host system. After the plug-in program exits, IMGW runtime puts the content of this file into the response event payload, then immediately unlinks this file.

hop_information_string: The string used to store the access information of the device. It is the string concatenation of all individual hop information of the device in order. An example the hop information and its <*hop_information_string*> are as follows:

Hop type	**IP address**	**Port**	**Username**	**Password**
IOS_LOGIN	172.29.145.45		Admin	Cisco
IOS_EN			Lab	Lab

The corresponding <*hop_information_string*> should be as follows:

```
"IOS_LOGIN" "172.29.145.45" " " "Admin" "Cisco" "IOS_EN" " " " " "Lab" "Lab"
```

Note For those fields of hop information with null value, IMGW runtime automatically adds a space before passing it to the child process.

command_to_be_executed: The command to be executed on the device. It is supplied by IMGW runtime only when the action type is **exec**.

command_arguments: The arguments of the command to be executed on the device. It is supplied by IMGW runtime only when the action type is **exec**.

configuration_file_name: The full path to the configuration file which will be downloaded onto the device. It is supplied by IMGW runtime only when the action type is **config** and its location is under */tmp* on the host system. After the plug-in program exits, IMGW runtime immediately unlinks this file.

persistence: y or **n**. The value **y** means the configuration needs to be written into non-volatile storage. It is supplied by IMGW runtime only when the action type is **config**. This option is dependent on the device type. This means the device module can ignore it if the device type does not support it.

operation_timeout_value: The maximum time period allowed to execute a command on the device. This parameter is now used by Expect scripts in IMGW legacy device module for IOS, CatOS, CatIOS, PIX, CSS and CE devices. A user-defined device module can ignore this parameter if it does not use it.

prompt_timeout_value: The maximum time period allowed to wait for the next prompt during login session to the device. This parameter is now used by Expect scripts in IMGW legacy device module for IOS, CatOS, CatIOS, PIX, CSS and CE devices. A user-defined device module can ignore this parameter if it does not use it.

Exit Codes

When the forked process (in which the plug-in program is executed) exits, the following exit codes are expected by IMGW runtime from the forked process:

config event:

0 – Download succeeds

1 – Download fails

2 – Download succeeds but with warning messages

Exec Event:

0 – Command execution succeeds

1 – Command execution fails

Hop Test:

0 – Hop test succeeds

1 – Hop test fails

How to Develop Plug-in Device Module

This toolkit allows the *Plug-in Developer* to use any implementation to realize the plug-in device module as long as the device module complies with IMGW southbound interface specified in "IMGW Southbound Interface" section on page 23-2.

This toolkit also provides sample code (see Toolkit Usage, page 23-1) in Perl plus Expect scripts as well as inline comments to help beginners to understand the workflow of the plug-in device module.

The plug-in device module should render three basic functions:

- Device configuration update
- Command execution
- Hop test

The first two functions are in response to the *cisco.mgmt.cns.config.load* and *cisco.mgmt.cns.exec.cmd* events respectively. The last one is an internal routine operation required by IMGW runtime and is transparent to network operators.

After IMGW runtime spawns a child process to execute the plug-in program, the corresponding device module should read the action type from the parameter list. If the action type is:

- **config** – device module should do device a configuration update.
- **exec** – device module should do a command execution.
- **hoptest –** device module should do hop test.

Development Guidelines

The following subsections describe the processes associated with each function.

Note The subject of actions in the subsections below is the plug-in device module.

Device Configuration Update

1. Access the device by way of the <*hop_information_string*>.
2. Download the configuration file named after <*configuration_file_name* > onto the device.
3. If the above download operation succeeds, the <*persistence*> is set to **y** and the device supports this option, then write the configuration to non-volatile storage.
4. Write all warning messages prompted by the device and their corresponding configuration commands line numbers into the file named after <*warning_logfile_name*> in the specified format (see "Parameter Descriptions" section on page 23-3). The content of this file will be part of the payload of the response event if the download succeeds but with warning messages.
5. Write all error messages prompted by the device and their corresponding configuration commands' line numbers into the file named after <*error_logfile_name*> in the specified format (see "Parameter Descriptions" section on page 23-3). The first error message and its corresponding configuration command line number will be part of the payload of the response event if the download fails.
6. Based on the <*logging_level*>, selectively redirect the processing history into the file named after <*temp_logfile_name*> for debugging purpose during the whole procedure.

7. Exit with proper exit code to return control to IMGW runtime. See "Exit Codes" section on page 23-5 to get the definition of exit codes.

Command Execution

1. Access the device by way of the <*hop_information_string*>.
2. Execute on the device the <*command_to_be_executed*> with the <*command_arguments*>.
3. Capture all output from the command execution into the file named after <*exec_response_logfile_name*>. The content of this file will be part of the payload of the response event.
4. Based on the <*logging_level*>, selectively redirect the processing history into the file named after <*temp_logfile_name*> for debugging purpose during the whole procedure.
5. Exit with proper exit code to return control to IMGW runtime. See "Exit Codes" section on page 23-5 to get the definition of exit codes.

Hop Test

1. Access the device by way of the <*hop_information_string*>.
2. Based on the <*logging_level*>, selectively redirect the processing history into the file named after <*temp_logfile_name*> for debugging purpose during the whole procedure.
3. Exit with proper exit code to return control to IMGW runtime. See "Exit Codes" section on page 23-5 to get the definition of exit codes.

Installing Plug-in Device Module

The *System Administrator* is required to take charge of the install/uninstall. He/She should make sure the installation is successful before calling the registration utility.

The System Administrator should install all plug-in device modules into the reserved file directory of */opt/ConfigEngine/CSCOimgw/plugin-modules* with one subdirectory per device module. For example, install the device module for MGX into */opt/ConfigEngine/CSCOimgw/plugin-modules/MGX* while install the one for NT into */opt/ConfigEngine/CSCOimgw/plugin-modules/NT*.

The *System Administrator* should only operate within the device module installation directory to set/remove the running environment of the module. The installation activities should not affect the running environment of other components on the Cisco Configuration Engine.

Registering Plug-in Device Module

The *System Administrator* must provide the device type and the full path to the plug-in program when registering a device module. IMGW runtime does not check the integrity of this information. It is responsibility of the *System Administrator* to make sure the information is correct.

This toolkit provides a dynamic registration utility to the system administrator, which allows the *System Administrator* to plug the device module into and out of IMGW seamlessly without tearing down IMGW runtime. Therefore, the services irrelevant to the device module that is being registered/de-registered will not be affected. However, this might not be the case for other services.

For example, at the time you issue the de-register command on device module *x*, the events related to *x* that are still queued in event bus might get failure responses from IMGW.

Caution It is HIGHLY RECOMMENDED that the *System Administrator* notify all *Network Operators* of the upcoming registration activities so that *Network Operators* have a chance to stop beforehand any relevant operation.

End User Interface

The end user interface of IMGW device module development toolkit consists of IMGW southbound interface and the command line registration utility.

Configuration and Restrictions

This toolkit does not put a restriction on the maximum number of plug-in device modules that can be put into IMGW.

Device Module Restrictions

- The device module must be able to run on the Linux and/or Solaris platform.
- If the executable of the device module is a C++ binary file, it must utilize the glib that exists on Cisco Configuration Engine where applicable.
- If the executable of the device module is a java class, it must run in the existing JVM of Cisco Configuration Engine.
- If the device module includes Perl and/or Expect scripts, the scripts should use the Perl and/or Expect interpreters that exist on Cisco Configuration Engine.

Registration Utility Restriction

The *System Administrator* is not allowed to register/de-register IMGW legacy device module. Sometimes users might want to modify one of the legacy device modules to do upload/download operation on CatOS, CatIOS, PIX, CSS, CE or IOS devices to meet their specific needs. In this case, they can only modify their own copy of the legacy device module, associate a different device type name to the modified device module and register the device module into IMGW.