Getting Started with Cisco UCS Director Open Automation

- Cisco UCS Director Open Automation, page 1
- Upgrading Your Connector to the Current Release, page 5
- Modules, page 5

Cisco UCS Director Open Automation

You can use the Cisco UCS Director Open Automation tools to develop and integrate your own Cisco UCS Director features as modules. You can customize modules to meet your unique needs.

Using the module, you can perform the following functions:

- Develop your own Cisco UCS Director reports and report actions
- Inventory your devices
- Track changes made to the system through your module
- Develop tasks that can be used for workflows
- Develop and schedule repeatable tasks
- Set up new resource limits

The Open Automation SDK bundle includes code samples that provide models, examples, and comments. You can download the SDK bundle with the sample code from Cisco DevNet.

Recommended Tools

We recommend that you use the following tools:

- Java version 1.8
- Eclipse (can be downloaded from www.eclipse.org)

Setting up Eclipse

Before You Begin

Install Java Runtime Environment (JRE) 1.8.

Step 1 In Eclipse, right-click the Cisco UCS Director Open Automation SDK and choose **Properties**.

Step 2 Set the **Java Compiler** to compile against 1.8.

Step 3 Click **OK**.

Note Make sure that you include the Cisco UCS Director Open Automation SDK jar files in your class path. Also make sure that your project setup mirrors the setup provided in the Open Automation SDK sample.

Downloading the Open Automation SDK Bundle

The Cisco UCS Director SDK binaries can be downloaded from the software download area of Cisco.com or the Cisco DevNet site. Also, an admin user can download the SDK binaries from Cisco UCS Director.

Step 1 Log in to Cisco UCS Director.

Step 2 On the menu bar, choose **Administration** > **Downloads**.

Step 3 Choose **Administration** > **Downloads**.

Step 4 Several files are displayed:

- REST API SDK—cucsd-rest-api-sdk-bundle.zip
- PowerShell Console—console.exe
- Open Automation SDK—cucsd-open-auto-sdk-bundle.zip
- Custom Tasks Script Samples—cucsd-cloupia-script-bundle.zip

Choose the **cucsd-open-auto-sdk-bundle.zip** file.

Step 5 Click **Download**.
The file is downloaded to the default download location.

Importing the Cisco UCS Director Open Automation SDK Project into the Eclipse IDE

The following instructions describe how to import the Open Automation SDK bundle into Eclipse. Follow the instructions provided for your development environment if you do not use Eclipse.

Step 1 Download the Open Automation SDK bundle from the Cisco.com download site or from Cisco DevNet.
Step 2 Extract the SDK bundle.
Step 3 Save the sample SDK project zip file on your file system.
Step 4 Launch Eclipse.
Step 5 Choose **File** > **Import**.
Step 6 In the **Import** dialog box, choose **General** > **Existing Projects into Workspace**.
Step 7 Click **Next**.
Step 8 Choose **Select root directory** and browse to the location where you extracted the project.
Step 9 Click **Finish**.
The project is automatically compiled.

Using EGit to Import the Open Automation SDK Bundle

Git with Eclipse (EGit) is an Eclipse plug-in that enables using the distributed version control system Git. EGit uses a connector plug-in in Eclipse to import the Open Automation SDK bundle into the IDE.

The Eclipse IDE downloaded from the www.Eclipse.org site contains support for Git in its default configuration. If the Git functionality is missing in your Eclipse IDE installation, you can use the Eclipse installation manager to install it. See the following:

- Installing the EGit Plug-In in Eclipse
- Importing the Open Automation SDK Bundles into Eclipse

Installing the EGit Plug-In in Eclipse

Step 1 Log in to Eclipse.
Step 2 Choose **Help** > **Install New Software**.
The **Install** window appears.
Step 3 Click the **Add** button available near the **Work with** field.

The **Add Site** window appears.

Step 4 Enter the repository location name.

Step 5 In the **Location** field, copy and paste the following URL: `http://download.eclipse.org/egit/updates/`.

Step 6 Click **OK** to add the repository location.
The Eclipse Git Team Provider and JGit packages appear.

Step 7 Check the **Eclipse Git Team Provider (Incubation)** check box.

Step 8 (Optional) Check the **JGit (Incubation)** check box.

Step 9 Click **Next**.
The chosen packages appear for verification.

Step 10 Click **Next**.

Step 11 Click **I accept the terms of the license agreement**.

Step 12 Click **Finish**.
All the necessary dependencies and executable are downloaded and installed.

Step 13 Accept the prompt to restart Eclipse.

What to Do Next

Import the open automation SDK bundle from the Git repository into Eclipse and run the SDK bundle.

Importing the Open Automation SDK Bundles into Eclipse

You can use Git to import the Cisco UCS DirectorSDK Bundles into Eclipse.

Before You Begin

You must have a Git account. If you do not have a Git account, sign up for a new account at GitLab.com.

Step 1 Log in to Eclipse.

Step 2 In the **Java** perspective, right-click in the **Package Explorer** pane.

Step 3 Click **Import**.

Step 4 Expand **Git**.

Step 5 Click **Projects from Git**.

Step 6 Click **Next**.

Step 7 Click **Clone URI**.

Step 8 In the **Import Projects from Git** window, perform the following operations:

a) In the **URI** field, enter the location of the source repository.
Enter one of the following source repository locations:

- The repository location for the sample code is https://gitlab.com/CUCSDSDK/OpenAutomationSample.git.
- The repository location for the storage module is https://gitlab.com/CUCSDSDK/OpenAutomationStorage.git.
- The repository location for the network module is https://gitlab.com/CUCSDSDK/OpenAutomationNetwork.git.
- The repository location for the compute module is https://gitlab.com/CUCSDSDK/OpenAutomationCompute.git.

The host and repository path are auto-populated.

b) In the **Authentication** section, enter your Git account credentials.
c) Click **Next**.
The branches available in the repository appear.
d) Check the branches that you want to clone from the Git repository.
e) Click **Next**.
f) In the **Directory** field, enter the local destination where you want to save the Open Automation SDK.
g) From the **Initial branch** drop-down list, choose **OpenAutomationSDK**.
By default, the remote name is set as **origin**.
h) Click **Finish**.

The Open Automation SDK bundle is imported into Eclipse. You can view the SDK bundle in the Project Explorer.

Upgrading Your Connector to the Current Release

Cisco UCS Director uses a connector to collect the inventory of the other vendor devices for managing those devices.
For more information on upgrading, refer the Cisco UCS Director Upgrade Guide.

Modules

A module is the top-most logical entry point into Cisco UCS Director.

A module can include the following components:

Component	Description
Task	A Workflow Task that can be used as part of a Workflow.
Report	A report that appears in the Cisco UCS Director UI. Reports may (but are not required to) contain clickable actions.
Trigger	A condition that, once satisfied, can be associated with some action. Examples: shutdown VM, start VM, and so on.

Guidelines for Developing a Module

When you develop a module to support new devices, follow these guidelines:

- Develop for a device family so that you have only one module to support all devices in the family.
- Develop a single module to support only devices within the same category. A module should handle only compute devices, network devices, or storage devices. For example, do not develop a module that

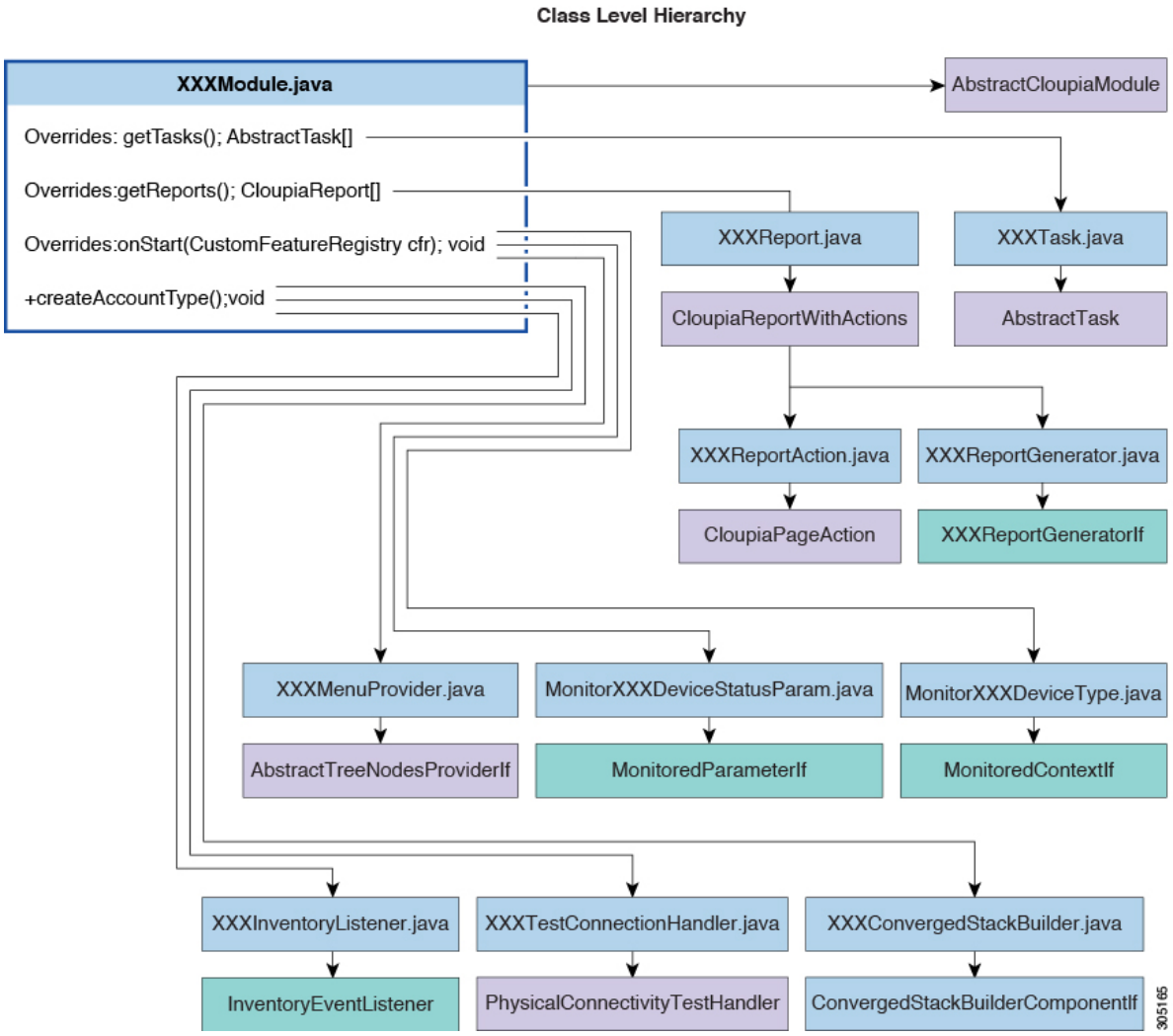
supports both a network switch and a storage controller. Instead, develop one module for the network switch and one module for the storage controller.

- Ensure that the devices supported by the same module are similar.
- The same device may come in different models that are meant for distinct purposes. In such cases, it may be appropriate to use different modules to support them.

Creating a Module

The following items must be in place for your custom module to work:

- A class extending `AbstractCloupiaModule`.
- Override the `OnStart` method in the Module Class that extends the `AbstractCloupiaModule`.
- A `.feature` file specifying your dependent jars and module class.
- A `module.properties` file is required in the custom module.



Before You Begin

Refer to **FooModule** in the sample project of the Open Automation SDK bundle.

Step 1 Extend the `AbstractCloupiaModule` class and register all your custom components in this class.

Step 2 Create a `.feature` file that specifies the dependent jars and module class.
This file must end with an extension of `.feature`; see foo.feature for reference. The best practice is to name this file with your module ID. For more details about the `.feature` file, see Packaging the Module, on page 10.

Step 3 Add the necessary custom jar files to the `lib` folder.

Step 4 Package the properties file at the root level of your module jar.
Cisco UCS Director provides you with a `properties` file for validation purposes. The SDK sample provides you with a build file that handles the packaging process.

Note The content of the `module.properties` file is described in Understanding the module.properties File, on page 8.

Step 5 In the `module.properties` file, replace the moduleID with the ID of the custom module.

Step 6 From the Eclipse IDE package explorer, right-click the `build.xml` file and run the ANT target build. This action generates the module `zip` file and save the file to the base directory of your project.

Understanding the module.properties File

The `module.properties` file exposes the module to the platform runtime. This file defines properties of the module.

Here is a sample `module.properties` file:

```
moduleID=foo
version=1.0
ucsdVersion=6.5.0.0
category=/foo
format=1.0
name=Foo Module
description=UCSD Open Automation Sample Module
contact=support@cisco.com
key=5591befd056dd39c8f5d578d39c24172
```

The contents are described in the following table:

Table 1: New Module.Properties (module.properties)

Name	Description
moduleID	The unique identifier for the module. This property is mandatory. Example: `moduleID=foo` **Tip** We recommend that you restrict this ID to a string of 3 to 5 lowercase alphabetic ASCII characters.
version	The current version of your module. This property is mandatory. Example: `version=1.0`
ucsdVersion	The version of Cisco UCS Director designed to support your module (with which your module works best). This property is mandatory. Example: `ucsdVersion=6.5.0.0`

Name	Description
category	The path (/location) where all your tasks must be placed. This property is mandatory. Example: `category=/foo` **Note** The category parameter is the full path to the location where your tasks are placed. If the tasks module is not validated, the path is set to `Open Automation Community Tasks/Experimental`. If the tasks module is validated, the tasks are placed relative to the root folder. For example, you can use `/Physical Storage Tasks/foo`, `/Open Automation Community Tasks/Validated/foo`, or `/foo`. In the last case, there is a folder at root level called `foo`. This feature enables developers to place tasks in categories that are not under Open Automation or in its categories.
format	The version of the format of this module. This property is mandatory. By default, 1.0 version is set for the custom module. Example: `format=1.0` **Restriction** `1.0` is the only acceptable value here.
name	A user-friendly string that identifies your module in the Open Automation reports. Example: `name=Foo Module`
description	A user-friendly description of what your module does. Example: `description=UCSD Open Automation Sample Module`
contact	An email address that consumers of your module can use to request support. Example: `contact=support@cisco.com`
key	An encrypted key that the Cisco UCS Director Open Automation group provides for validating the module. Example: `key=5591befd056dd39c8f5d578d39c24172`

Note Modifying any mandatory properties invalidates your module. If you change any of the mandatory properties, you must request validation again. The name, description, and contact values, which are not mandatory, can be modified or omitted without revalidation.

Packaging the Module

A module is packaged with all the necessary classes, dependent JAR files, a `module.properties` file, and a `.feature` (pronounced "dot-feature") file. The `.feature` file is placed in the same folder as the root of the project. The `.feature` file shows the JAR associated with this module and the path to the dependent JAR files. The name of the `.feature` file is `<moduleID>-module.feature`.

The following example shows the content of a `.feature` file:

```
{
      jars:  [ "features/feature-chargeback.jar",
      "features/chargeback/activation-1.1.jar",
      "features/chargeback/axis2-jaxbri-1.5.6.jar",
      "features/chargeback/bcel-5.1.jar",
      "features/chargeback/jalopy-1.5rc3.jar",
      "features/chargeback/neethi-2.0.5.jar",
      "features/chargeback/antlr-2.7.7.jar",
      "features/chargeback/axis2-jaxws-1.5.6.jar",]
      features: [ "com.cloupia.feature.oabc.OABCModule" ]
}
```

Before You Begin

We recommend that you use the Apache ANT build tool that comes with Eclipse. You can use any build tool or create the build by hand, but you must deliver a package with the same characteristics as one built with ANT.

SUMMARY STEPS

1. If your module depends on JARs that are not provided with the sample source code, include the jars in the `build.xml` file so that they are packaged in the zip file.
2. From the `build.xml` file, run the ANT target build.

DETAILED STEPS

	Command or Action	Purpose

Step 1	If your module depends on JARs that are not provided with the sample source code, include the jars in the `build.xml` file so that they are packaged in the zip file.	The following example shows a module layout with a third-party JAR:
		`feature-oabc`
		`feature oabc.jar`
		`oabc`
		`lib`
		`flex`
		`flex-messaging-common.jar`
		`oabc.feature`

	Command or Action	Purpose

		The module jar and `.feature` are at the top level of the zip file. We recommend that you put the third-party jars under the `/moduleID/lib` folder path, then any other sub-directories you may want to add.
		`{`
		`jars: [ 'features/feature-oabc.jar", features/oabc/lib/flex-messaging-common.jar ],`
		`features: [ "com.cloupia.feature.oabc.OABCModule" ]`
		`}`
		When you list the jars in the `.feature` file, ensure that the jars start with `features/`; this is mandatory. This convention enables you to include the path to the jar. The path of each jar must be the same path that is used in your zip file. We recommend that you put your module jar first, followed by its dependencies, to ensure that your module loads.

Step 2	From the `build.xml` file, run the ANT target build.	The zip file is generated and saved to the base directory of your project. (We recommend that you create your own project directory for your module. For convenience, in this example we assume that the sample project is the base directory for your project.)

Deploying a Module on Cisco UCS Director

The Cisco UCS Director user interface provides **Open Automation** controls that you can use to upload and manage modules. Use these controls to upload the `zip` file of the module to Cisco UCS Director.

Note Only `zip`-formatted files can be uploaded using the **Open Automation** controls.

Before You Begin

Acquire shell administrator access on the Cisco UCS Director VM. You can get this access from your system administrator. To use the Cisco UCS Director Shell Menu as a shell administrator, use SSH to access Cisco UCS Director, using the login **shelladmin** with the password that you got from the administrator.

For SSH access in a Windows system, use PuTTY (see http://www.putty.org/). On a Mac, use the built-in terminal application's SSH utility.

Step 1 Choose **Administration** > **Open Automation**.

Step 2 On the **Open Automation** page, click **Modules**.
The **Modules** page displays the following columns:

Column	Description
ID	The ID of the module.

Column	Description
Name	The name of the module.
Description	The description of the module.
Version	The current version of the module. The module developer must determine how to administer versioning of the module.
Compatible	Which version of Cisco UCS Director best supports this module.
Contact	The contact information of the person responsible for technical support for the module.
Upload Time	The time at which the module was uploaded.
Status	The status of the module. Possible statuses are: Enabled, Disabled, Active, and Inactive. You can control whether a module is enabled or disabled. If enabled, Cisco UCS Director attempts to initialize the module; if disabled, Cisco UCS Director ignores the module. A module is set to the Active state only when Cisco UCS Director is able to successfully initialize the module without throwing an exception. **Note** Active does not necessarily mean that everything in the module is working properly; it merely indicates that the module is up. Inactive means that when Cisco UCS Director tried to initialize the module, a severe error prevented it from doing so. Typical causes for the Inactive flag are: the module is compiled with the wrong version of Java, or a class is missing from the module.
Validated	Indicates whether the module is validated or not.

Note To enable module activation on upload, ensure that the `.feature` file in your module is named after your module ID. For example: If moduleId is `myFeatureName`, then name your feature file `myFeatureName.feature`.

The Cisco UCS Director framework identifies and loads the `.feature` file by name, based on the module ID. If the name of the `.feature` file and the module ID are different, the `.feature` file does not load and the module is not activated. If you choose to give the module ID and the .feature file different names, you must restart Cisco UCS Director to activate the module.

Step 3 Click **Add** to add a new module.

The **Add Modules** dialog box appears.

Step 4 Choose the module zip file from your local files and click **Upload** to upload the module zip file.

Step 5 Enable the module by choosing the module in the **Modules** table and clicking **Enable**.

Step 6 Wait while Cisco UCS Director activates the module.

Note Restarting Cisco UCS Director is not required to enable, disable, delete, or modify a module.

What to Do Next

Once the module is active, you can test the module.

Deleting, Deactivating, and Modifying Connectors

In addition to creating new connectors, you can also:

- Disable connectors
- Modify connectors (install new connector versions)
- Delete connectors from Cisco UCS Director

In previous releases, Cisco UCS Director had to be restarted for these changes to take effect. This restart is no longer required.