



# 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 you to use the following tools:

- Java version 1.8
- Eclipse (can be downloaded from [www.eclipse.org](http://www.eclipse.org))

## Setting up Eclipse

### Before You Begin

Java Runtime Environment (JRE) 1.8 is required.

- 
- Step 1** In Eclipse, click the Cisco UCS Director Open Automation SDK. Right-click and choose **Properties**.
- Step 2** Set the **Java Compiler** to compile against 1.8 and 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 or the [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**.  
The following 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
- Step 3** Choose the **cucsd-open-auto-sdk-bundle.zip** file and 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.

- 
- |               |   |
|---------------|---|
| <b>Step 1</b> | Download the Open Automation SDK bundle from the <a href="#">Cisco.com download site</a> or from <a href="#">Cisco DevNet</a> . Extract the SDK bundle and save the sample SDK project zip file in to your file system. |
| <b>Step 2</b> | Launch Eclipse.   |
| <b>Step 3</b> | Choose <b>File &gt; Import</b> .  |
| <b>Step 4</b> | In the <b>Import</b> dialog box, choose <b>General &gt; Existing Projects into Workspace</b> .  |
| <b>Step 5</b> | Click <b>Next</b> .   |
| <b>Step 6</b> | Choose <b>Select root directory</b> and browse to the location where you extracted the project.   |
| <b>Step 7</b> | Click <b>Finish</b> .<br>The project is automatically compiled.   |
- 

## Using EGit to Import the Open Automation SDK Bundle

Git with Eclipse (EGit) is an Eclipse plug-in to use the distributed version control system Git. EGit develops a connector plug-in in IDE to import the open automation SDK bundle into the IDE.

Most Eclipse IDE downloaded from the [www.eclipse.org](http://www.eclipse.org) site contains support for Git in their default configuration. If the Git functionality is missing in your Eclipse IDE installation, you can install it through the Eclipse installation manager. For more information on how to install the EGit plug-in, see the [Installing EGit Plug-In in Eclipse](#), on page 3.

- [Installing EGit Plug-In in Eclipse](#)
- [Importing the Open Automation SDK Bundles into Eclipse](#)

### Installing EGit Plug-In in Eclipse

- 
- |               |  |
|---------------|--|
| <b>Step 1</b> | Log in to Eclipse.   |
| <b>Step 2</b> | Choose <b>Help &gt; Install New Software</b> .<br>The <b>Install</b> window appears. |
| <b>Step 3</b> | Click the <b>Add</b> button available near the <b>Work with</b> 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 package appears.
- 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 packages that are chosen for installation 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

### Before You Begin

Ensure that you have a Git account. If you do not have a Git account, signup for a new account in GitLab.com.

- 
- Step 1** Log in to Eclipse.
- Step 2** In the Java perspective, right-click in the **Package Explorer** pane and click **Import**.
- Step 3** Expand **Git** and click **Projects from Git**.
- Step 4** Click **Next**.
- Step 5** Click **Clone URI**.
- Step 6** In the **Import Projects from Git** window, perform the following operations:
- In the **URI** field, enter the location of the source repository.  
Enter one of the following source repository locations according to the requirement:
    - Repository location for the sample code is <https://gitlab.com/CUCSDSDK/OpenAutomationSample.git>.
    - Repository location for the storage module is <https://gitlab.com/CUCSDSDK/OpenAutomationStorage.git>.
    - Repository location for the network module is <https://gitlab.com/CUCSDSDK/OpenAutomationNetwork.git>.
    - Repository location for the compute module is <https://gitlab.com/CUCSDSDK/OpenAutomationCompute.git>.
- The host and repository path are autopopulated.
- In the **Authentication** section, enter the Git account credential.

- c) Click **Next**.

The branches available in the repository appears.

- d) Check the check box of 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 while defining a Workflow.
Reports	Reports that appear in the Cisco UCS Director UI. Reports may or may not contain action buttons.
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 new module to support new devices, ensure that:

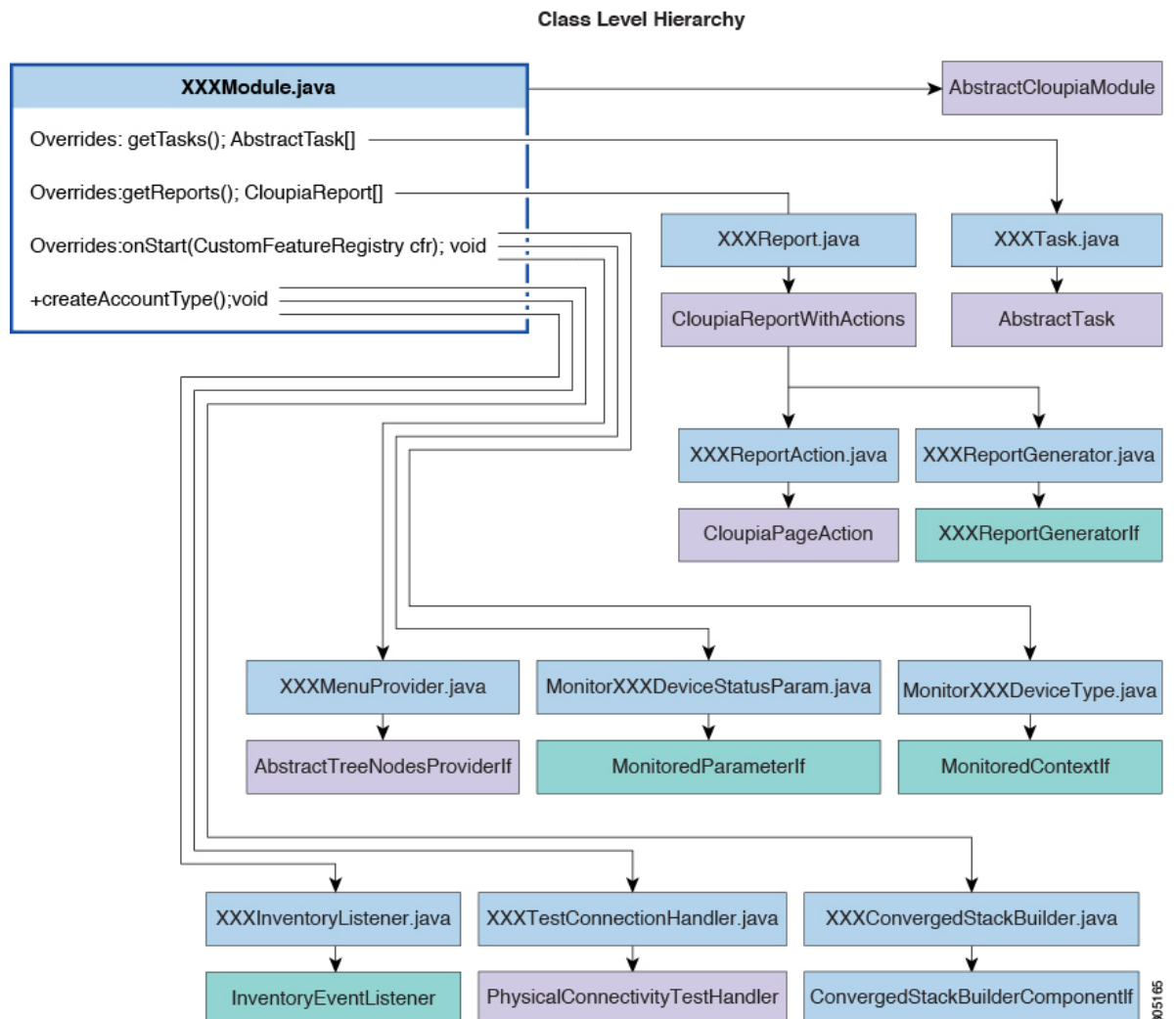
- You develop a module for a device family so that you have only one module to support all these devices.
- You do not develop a module that supports both a network switch and a storage controller; instead, split them into two modules. Ideally, a module must support only devices within the same category, so that a module can handle only compute devices, network devices, or storage devices.
- The devices supported by the same module must be similar.

- The same device may come in different models that are meant for distinct purposes, and 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 9.
- 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 7.
- 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 certain properties of the module.

Here is a sample `module.properties` file:

```
moduleID=foo
version=1.0
ucsdVersion=5.4.0.0
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	<p>The unique identifier for the module. This property is mandatory.</p> <p>Example:</p> <pre>moduleID=foo</pre> <p><b>Tip</b> The recommended best practice is to restrict this ID to a string of 3 to 5 lowercase ASCII alphabet characters.</p>

Name	Description
version	The current version of your module. This property is mandatory. Example: <code>version=1.0</code>
ucsdVersion	The version of Cisco UCS Director designed to support your module (with which your module works best). This property is mandatory. Example: <code>ucsdVersion=5.4.0.0</code>
category	The path (/location) where all your tasks must be placed. This property is mandatory. Example: <code>category=/foo</code> <b>Note</b> Category is the full path to the location where your tasks are placed. If the tasks module is not validated, the path is set under Open Automation Community Tasks/Experimental. If the tasks module is validated, the tasks are placed anywhere relative to the root folder. For example, you can use /Physical Storage Tasks/foo, so that, the tasks are under that folder or under /Open Automation Community Tasks/Validated/foo or under /foo. In theory, for this last case, there is a folder at root level called foo. This change 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: <code>format=1.0</code> <b>Restriction</b> As of Cisco UCS Director Release 5.0.0.0, "1.0" is the only acceptable value here.
name	A user-friendly string that is used to identify your module in the open automation reports. Example: <code>name=Foo Module</code>
description	The user-friendly text that describes what your module does. Example: <code>description=UCSD Open Automation Sample Module</code>
contact	An email address that consumers of your module can use to request support. Example: <code>contact=support@cisco.com</code>



Name	Description
key	<p>An encrypted key that the Cisco UCS Director Open Automation group provides for validating the module.</p> <p>Example:</p> <pre>key=5591befd056dd39c8f5d578d39c24172</pre>

**Note**

If you attempt to modify the mandatory properties, the updates make your module invalidate. If you change any of the mandatory properties, you must request validation again. In contrast, 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 dependent JAR files, classes, and a `module.properties` file along with a `.feature` file. The `.feature` (pronounced "dot-feature") file is placed in the same folder as the root of the project. This 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`.

We recommend you to use the Apache ANT™ build tool that comes with Eclipse. You can also use any other build tool or create the build by yourself, but you have to deliver a package with the same characteristics as one built with ANT.

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" ]
}
```

From the `build.xml` file, run the ANT target build. This action generates the necessary zip file and save it to the base directory of your project. (This assumes that you are using the sample project as the base of your own project. Although using the sample project in this way is not recommended, it is the basis of this demonstration.)

If your module depends on JARs that are not provided with the sample source code, include the jars in the `build.xml` file to have them 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
```

The module jar and `.feature` are at the top level of the zip file. Place the third-party jars under the `/moduleID/lib/` folder path. Although it is not required, the best practice is to place 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" ]
}
```

References to the jar files must always start with `features/`. When you list the jars in the `.feature` file, ensure that the jars start with `features/`. This action 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. The best practice is to lead with your module jar, followed by its dependencies, to ensure that your module gets loaded.

## Deploying the 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

For uploads, only the zip file format is supported.

### Before You Begin

To enable or activate a module, restart the Cisco UCS Director services, for which you require **shell admin** access. You can get this access from your system administrator. To use the **Cisco UCS Director Shell Menu** as a shell administrator, you have to use SSH to access Cisco UCS Director, using the login `shelladmin` with the password that you got from the administrator. For the SSH access in a Windows system, use PuTTY; (see <http://www.putty.org/>); on a Mac, use the built-in SSH utility.

### Step 1

In Cisco UCS Director, choose **Administration > Open Automation**.

The **Modules** table appears and displays the following columns:

Column	Description
<b>ID</b>	The ID of the module.
<b>Name</b>	The name of the module.
<b>Description</b>	The description of the module.
<b>Version</b>	The current version of the module. The module developer must determine how to administer versioning of the module.
<b>Compatible</b>	Displays which version of Cisco UCS Director best supports this module.
<b>Contact</b>	The contact information of the person responsible for technical support for the module.
<b>Upload Time</b>	The time at which the module is uploaded.

Column	Description
Status	<p>The status of the module. The status includes: Enable, Disable, Active, and Inactive.</p> <p>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 exception.</p> <p><b>Note</b> 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 not included in the module.</p>
Validated	Indicates whether the module is validated or not.

- Step 2** Click **Add** to add a new module.  
The **Add Modules** dialog box appears.
- Step 3** Choose the module zip file from your local files and click **Upload** to upload the module zip file.
- Step 4** Enable the module by choosing the module in the **Modules** table and clicking **Enable**.
- Step 5** Activate the module by restarting Cisco UCS Director.
- Step 6** In Cisco UCS Director, navigate to **Administration > Open Automation** and verify that the module status is Active.
-

