Managing Modules

This chapter contains the following sections:

Modules

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

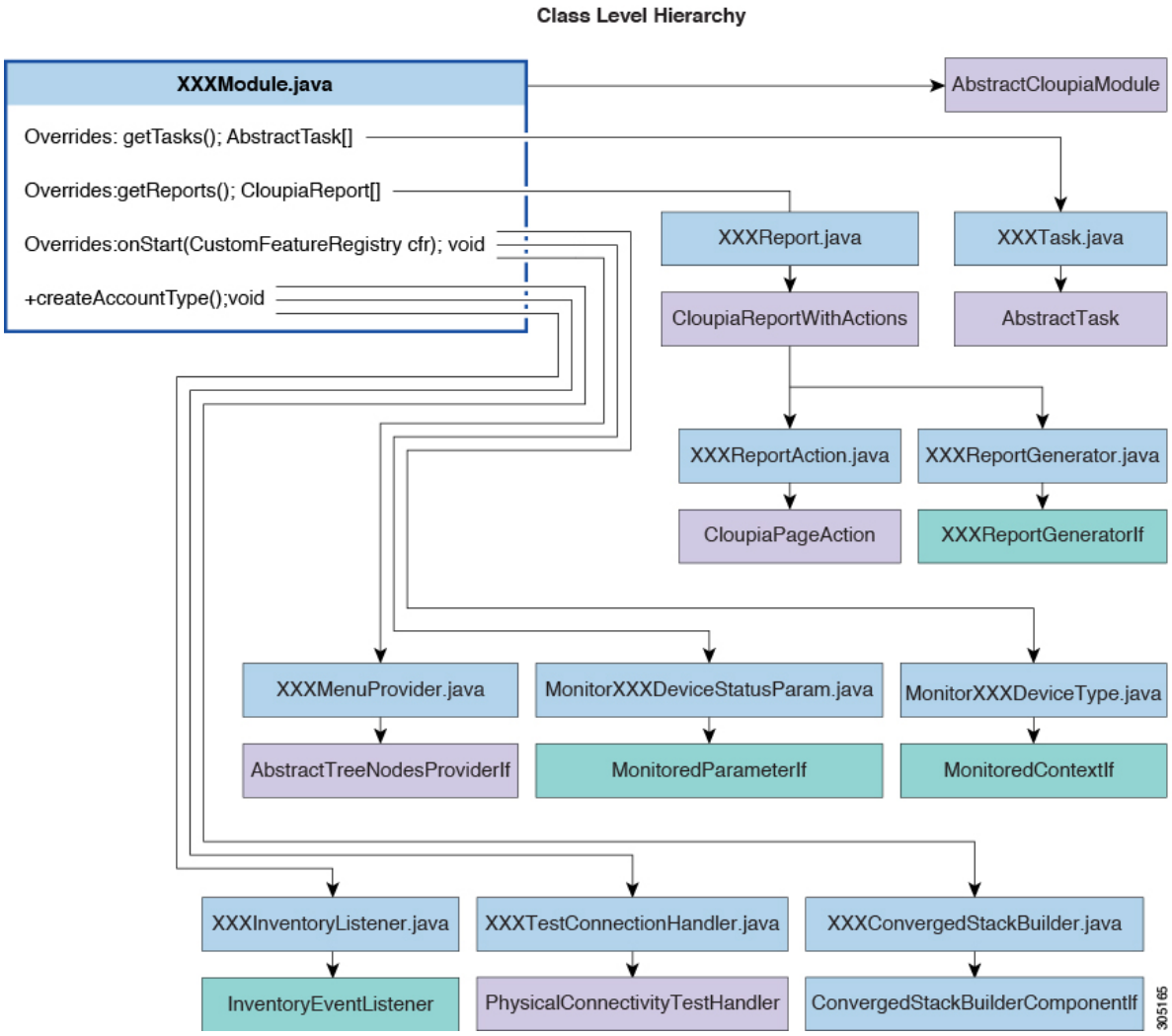
A module can include the following components:

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.

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.

Procedure

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 5.

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 3.

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	The unique identifier for the module. This property is mandatory. Example: `moduleID=foo` **Tip** The recommended best practice is to restrict this ID to a string of 3 to 5 lowercase ASCII alphabet 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=5.4.0.0`

Name	Description
category	The path (/location) where all your tasks must be placed. This property is mandatory. Example: `category=/foo` **Note** 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: `format=1.0` **Restriction** 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: `name=Foo Module`
description	The user-friendly text that describes 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 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.

Procedure

Step 1 In Cisco UCS Director, choose **Administration** > **Open Automation**.
The **Modules** table appears and displays the following columns:

Column	Description
ID	The ID of the module.
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	Displays 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 is uploaded.

Column	Description
Status	The status of the module. The status includes: Enable, Disable, 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 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 not included in the module.
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.

Deactivating a Module

To deactivate a module you must stop and restart the Cisco UCS Director services for your change to take effect.

Procedure

Step 1 Choose the module you need to deactivate in the **Modules** table, then click the **Deactivate** control.

Step 2 Stop and restart the Cisco UCS Director services. Follow the same procedure that you use after activating a module.