CONTENTS

**Preface** vii

Audience vii

Conventions vii

Related Documentation ix

Documentation Feedback ix

Obtaining Documentation and Submitting a Service Request ix

---

**CHAPTER 1**

New and Changed Information for this Release 1

New and Changed Information 1

---

**CHAPTER 2**

Overview 3

About Cisco UCS Director Development Platform Architecture 3

About Cisco UCS Director Open Automation 3

Recommended Tools 4

---

**CHAPTER 3**

Getting Started 5

Getting the Latest Cisco UCS Director Open Automation SDK 5

Setting Up Eclipse 5

Verifying Your Setup 6

---

**CHAPTER 4**

Developing and Deploying Modules 7

About Modules 7

Creating a Module 8

Publish the Module 8

Creating Tasks 9

Packaging the Module 9

Deploying the Module on Cisco UCS Director 10
Deactivate Module 10

CHAPTER 5 Creating a New Module 13
Creating a New Module 13

CHAPTER 6 Working with a Module 15
About the Module and Its Components 15
Object Store 16
  Marking a Class for Persistence 16
  Publishing the Persistence Class 17
  Performing CRUD Operations on the Persistence Class 17
Annotations 18
Lists of Values (LOVs) 18
  Defining Your Own List Provider 18
Tables (Tabular Reports) 19
Tasks 19

CHAPTER 7 Adding a New Workflow Task to a Module 21
About Workflow Tasks 21
Developing a TaskConfigIf 21
Developing an Abstract Topic 23
Registering Custom Workflow Inputs 24
Registering Custom Task Output 24
Consuming Custom Output as Input in Other Tasks 25
Consuming Output from Existing Tasks as Input 25
Verifying the Custom Task Is In Place 26
Debugging Tasks 27

CHAPTER 8 Developing Your Own Inventory Collector 29
About the Inventory Collector 29
Guidelines for Developing a Module 30
Connecting to the Device (NodeConnectorIf) 31
Binding Collected or Parsed Data to POJO (ItemDataObjectBinderIf) 31
Using Generic Infra Account Report to Simplify Account Management 31
Registering Inventory Collection Components and Account Management Reports 32
CHAPTER 14

Developing Report Actions  61
   About Report Actions   61
   Registering Actions with Report  62

CHAPTER 15

Developing CloudSense Reports  63
   Developing CloudSense Reports  63

CHAPTER 16

Change Tracking API  65
   Change Tracking API  65

APPENDIX A

Appendix A  67
   Existing List of Value Tables  67

APPENDIX B

Appendix B  71
   Report Context Types and Report Context Names  71
Preface

This preface contains the following sections:

- Audience, page vii
- Conventions, page vii
- Related Documentation, page ix
- Documentation Feedback, page ix
- Obtaining Documentation and Submitting a Service Request, page ix

Audience

This guide is intended for software engineers with expertise using APIs to develop and extend applications. These engineers should understand Cisco UCS and related networking and storage protocols, and have experience working with JSON, XML, and Java.

Conventions

<table>
<thead>
<tr>
<th>Text Type</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI elements</td>
<td>GUI elements such as tab titles, area names, and field labels appear in this font. Main titles such as window, dialog box, and wizard titles appear in this font.</td>
</tr>
<tr>
<td>TUI elements</td>
<td>In a Text-based User Interface, text the system displays appears in this font.</td>
</tr>
<tr>
<td>System output</td>
<td>Terminal sessions and information that the system displays appear in this font.</td>
</tr>
<tr>
<td>CLI commands</td>
<td>CLI command keywords appear in this font. Variables in a CLI command appear in this font.</td>
</tr>
<tr>
<td>Text Type</td>
<td>Indication</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>[ ]</td>
<td>Elements in square brackets are optional.</td>
</tr>
<tr>
<td>{x</td>
<td>y</td>
</tr>
<tr>
<td>[x</td>
<td>y</td>
</tr>
<tr>
<td>string</td>
<td>A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.</td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td>Nonprinting characters such as passwords are in angle brackets.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Default responses to system prompts are in square brackets.</td>
</tr>
<tr>
<td>!, #</td>
<td>An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.</td>
</tr>
</tbody>
</table>

**Note**

Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the document.

**Tip**

Means *the following information will help you solve a problem*. The tips information might not be troubleshooting or even an action, but could be useful information, similar to a Timesaver.

**Caution**

Means *reader be careful*. In this situation, you might perform an action that could result in equipment damage or loss of data.

**Timesaver**

Means *the described action saves time*. You can save time by performing the action described in the paragraph.

**Warning**

IMPORTANT SAFETY INSTRUCTIONS

This warning symbol means danger. You are in a situation that could cause bodily injury. Before you work on any equipment, be aware of the hazards involved with electrical circuitry and be familiar with standard practices for preventing accidents. Use the statement number provided at the end of each warning to locate its translation in the translated safety warnings that accompanied this device.

SAVE THESE INSTRUCTIONS
Related Documentation

Cisco UCS Director Documentation Roadmap

For a complete list of Cisco UCS Director documentation, see the Cisco UCS Director Documentation Roadmap available at the following URL: http://www.cisco.com/en/US/docs/unified_computing/ucs/ucs-director/doc-roadmap/b_UCSDirectorDocRoadmap.html.

Cisco UCS Documentation Roadmaps

For a complete list of all B-Series documentation, see the Cisco UCS B-Series Servers Documentation Roadmap available at the following URL: http://www.cisco.com/go/unifiedcomputing/b-series-doc.

For a complete list of all C-Series documentation, see the Cisco UCS C-Series Servers Documentation Roadmap available at the following URL: http://www.cisco.com/go/unifiedcomputing/c-series-doc.

Note

The Cisco UCS B-Series Servers Documentation Roadmap includes links to documentation for Cisco UCS Manager and Cisco UCS Central. The Cisco UCS C-Series Servers Documentation Roadmap includes links to documentation for Cisco Integrated Management Controller.

Documentation Feedback

To provide technical feedback on this document, or to report an error or omission, please send your comments to ucs-director-docfeedback@cisco.com. We appreciate your feedback.

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly What's New in Cisco Product Documentation, which also lists all new and revised Cisco technical documentation.

Subscribe to the What's New in Cisco Product Documentation as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS version 2.0.
New and Changed Information

The following table provides an overview of the significant changes to this guide for this current release. The table does not provide an exhaustive list of all changes made to this guide or of all new features in this release.

Table 1: New Features and Changed Behavior in Cisco UCS Director Open Automation, Release 4.1.0.3

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Where Documented</th>
</tr>
</thead>
<tbody>
<tr>
<td>New support for building bar chart reports</td>
<td>Instructions, sample code, and support to enable developers to create bar charts in Cisco UCS Director.</td>
<td>Developing Bar Chart Reports, on page 45</td>
</tr>
<tr>
<td>New support for building line chart reports</td>
<td>Instructions, sample code, and support to enable developers to create line charts in Cisco UCS Director.</td>
<td>Developing Line Chart Reports, on page 46</td>
</tr>
<tr>
<td>New support for building pie chart reports</td>
<td>Instructions, sample code, and support to enable developers to create pie charts in Cisco UCS Director.</td>
<td>Developing Pie Chart Reports, on page 47</td>
</tr>
<tr>
<td>New support for building heat map reports</td>
<td>Instructions, sample code, and support to enable developers to create heat map reports in Cisco UCS Director.</td>
<td>Developing Heat Map Reports, on page 48</td>
</tr>
<tr>
<td>New support for building summary reports</td>
<td>Instructions, sample code, and support to enable developers to create summary reports in Cisco UCS Director.</td>
<td>Developing Summary Reports, on page 49</td>
</tr>
<tr>
<td>New support for building forms as reports</td>
<td>Instructions, sample code, and support to enable developers to create Config Forms in Cisco UCS Director.</td>
<td>Developing Form Reports, on page 51</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Where Documented</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>New support for adding trigger conditions for monitoring purposes.</td>
<td>Instructions, sample code, and support to enable developers to create new trigger conditions in Cisco UCS Director.</td>
<td>Developing a Trigger Condition, on page 55</td>
</tr>
<tr>
<td>New support for adding menu items in the UI</td>
<td>Instructions, sample code, and support to enable developers to create and add new menu items in Cisco UCS Director.</td>
<td>Developing a New Menu, on page 35</td>
</tr>
</tbody>
</table>
Overview

This chapter contains the following sections:

- About Cisco UCS Director Development Platform Architecture, page 3
- About Cisco UCS Director Open Automation, page 3
- Recommended Tools, page 4

About Cisco UCS Director Development Platform Architecture

The Cisco UCS Director platform architecture consists of two components: the Cisco UCS Director Platform Runtime and the set of modules that get executed on the platform. The platform provides the required APIs and management functionality so that the modules can perform their intended functions. The modules provide the necessary intelligence and features. The platform provides a loosely-coupled plug-in architecture where modules can be developed, deployed and executed against the platform.

Using tools such as Cisco UCS Director Open Automation and the SDK, you can develop and deploy a module that can be executed on the Cisco UCS Director platform runtime.

About 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, enhance its functionality and customize modules to meet your unique needs.

Using Cisco UCS Director Open Automation to modify UCS Director modules, you can:

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

Recommended Tools

The tasks discussed in this document assume that you are using these tools:

Java version 1.6


Restriction

Other releases, such as Java 1.7, are not supported.

Eclipse

Getting Started

This chapter contains the following sections:

- Getting the Latest Cisco UCS Director Open Automation SDK, page 5
- Setting Up Eclipse, page 5
- Verifying Your Setup, page 6

Getting the Latest Cisco UCS Director Open Automation SDK

You can download or request the latest Cisco UCS Director Open Automation SDK from Cisco. The Open Automation SDK includes code samples that provide models, examples and comments. This documentation refers frequently to these samples.

You can find the Open Automation SDK with all other Cisco UCS Director software in the download area on Cisco.com.

Setting Up Eclipse

The Cisco UCS Director team recommends that you use Eclipse IDE for development. Click here to go to www.eclipse.org.

**Before You Begin**

Java Runtime Environment (JRE) 1.6 is required.

| Restriction | Other releases, such as Java 1.7, are not supported. |

**Procedure**

1. In Eclipse, go to the Properties for the Cisco UCS Director Open Automation SDK.
2. Set the Java Compiler to compile against 1.6 and click **OK**.
Verifying Your Setup

You can test your development environment by importing the sample SDK project into your Eclipse workspace.

Note

The samples provided are intended to work with the module.properties file with which they are bundled. To avoid runtime exceptions and module loading issues, replace all the sample classes with your own implementations.

Procedure

Step 1 You can find the Open Automation SDK with all other Cisco UCS Director software in the download area on Cisco.com. To download the SDK, click on the link to the UCS Director SDK. Unpack the sample SDK project zip file in your file system.

Step 2 Launch Eclipse, and go to File > Import.

Step 3 In the Import dialog box, choose General > Existing Projects into Workspace, then click Next.

Step 4 In the Import Projects dialog box, choose Select root directory, and browse to the location where you unpacked the project, then click Finish. The project automatically compiles.
Developing and Deploying Modules

This chapter contains the following sections:

- About Modules, page 7
- Creating a Module, page 8
- Publish the Module, page 8
- Creating Tasks, page 9
- Packaging the Module, page 9
- Deploying the Module on Cisco UCS Director, page 10

About Modules

A module is the top-most logical entry point into Cisco UCS Director. To add or extend any functionality, you must develop and deploy a module on Cisco UCS Director.

To develop and deploy a module, complete the following procedures:

- Create a module by implementing a class that extends the AbstractCloupiaModule class.
- Add your custom tasks, and reports.
- As necessary, create tasks by implementing the necessary task interfaces.
- Package the module. (Recommendation: Use Apache ANT™ or Apache Maven™ build tools.)
- Deploy the package on Cisco UCS Director

The instructions provided here assume you are using the ANT build tool that comes with Eclipse. You may instead use the Apache Maven build tool or create the build yourself, but you will have to deliver a package with the same characteristics as one built with ANT.

These procedures are explained in separate topics.
Creating a Module

A module class extends the `AbstractCloupiaModule` class and provides implementation for the single method `getTasks()`. This comment on the `getTask()` method explains the fundamental requirements:

```java
/**
 * External modules extending this class shall provide implementation for this method.
 * @return an array of tasks supported by this module
 */
```

Adding tasks to a module is a key goal. See About Workflow Tasks, on page 21, for more detailed instructions about adding a workflow task to a module.

Publish the Module

In order to expose the module to the Platform Runtime, a file named `module.properties` is provided along with the module. This properties file defines certain properties of the module itself.

Here is a sample `module.properties` file:

```properties
featureID=oaabc
status=experimental
org=cisco
suborg=cloupia
feature=something
key=abcdef
```

The contents are as follows:

- **moduleID** — A unique readable string defining the feature.
- **status** — Experimental or Validated. A developer version of the module is always "experimental". If the module is validated by the UCS Director team, the status is "validated".
- **categoryID** — A unique string to categorize the module's tasks.
- **categoryLabel** — A user friendly string to accompany the categoryID.
- **org** — Defines the name of the organization to which this module belongs.
- **Sub-org** — The name of the suborganization to which this module belongs.
- **Feature** — For future use.
- **Key** — For future use.
Creating Tasks

Creating tasks is one of the more important procedures you can perform when working with a module, but the details involved in this procedure require in-depth treatment, and are therefore covered in a more advanced section: About the Module and Its Components, on page 15.

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.

This example shows a .feature file. The name of the file will be <moduleID>-module.feature

```
{
    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, run the ANT target build. This will generate the necessary Zip file and save it to the base directory of your project. (This assumes 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 is dependent on JARs not provided with the sample source code, you must include them in the build.xml so they will be included in the Zip file. This example shows a module layout containing a third-party JAR:

```
feature-oabc
    feature oabc.jar
    oabc
        lib
            flex
                flex-messaging-common.jar
    oabc.feature
```

Note that the module jar and .feature are at the top level of the zip file. Then you place your third-party jars under /moduleID/lib/. Although it is not required, the best practice is to place your third-party jars under /moduleID/lib, 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.OABC(Module" ]
}
```

Note that references to the jar files always start with features/. This is mandatory. In your .feature file, when you list out the jars, it starts with features/. This enables you to include the path to the jar you want. The path of each jar should 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 for the module to Cisco UCS Director.

---

**Note**

For uploads, only the Zip file format is supported.

---

**Before You Begin**

To deploy or deactivate a module, you must restart Cisco UCS Director services, for which you will require **shell admin** access. You should be able to get this access from your system administrator. To use the **Cisco (CUIC) Shell Menu** as a shell administrator, you will need to use SSH to access Cisco UCS Director, using the login shelladmin with the password that your administrator provides. For the SSH access in a Windows system, use PuTTY; (see [http://www.putty.org/](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.

**Step 2**

Select **Add** to add a new module. The **Add Modules** dialog box appears, enabling you to provide the module name, the Zip file to upload, and a module description.

**Step 3**

Provide the required information and click **Submit**.

**Important** To load the newly added module, you must make the module active, and restart Cisco UCS Director services after the upload.

**Step 4**

Activate the Module by choosing the module in the **Modules** table, then click Add. The **Activate Module** dialog box appears.

**Step 5**

Click **Submit** to activate the module. In the event of an error, a clear error message will appear, indicating why the module was not acceptable.

**Step 6**

Stop and restart Cisco UCS Director services.

a) Open the **Shell Menu** with your shell admin access. The **Cisco (CUIC) Shell Menu** opens, prompting you to "Select a number from the menu below".

b) At the SELECT prompt, enter 3 to stop services. Then enter 4 to start services again.

It should take Cisco UCS Director a couple of minutes to restart. Once the system is ready, your module should be loaded and your tasks visible in the Cisco UCS Director GUI.

---

**Deactivate Module**

As when activating a module, to deactivate a module you must stop and restart the Cisco UCS Director services in order for your change to take effect.

To deactivate a module:
**Deactivate Module**

**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.
Deactivate Module
Creating a New Module

This chapter contains the following sections:

- Creating a New Module, page 13

Creating a New Module

Four items need to be in place for your custom module to work:

- A class extending AbstractCloupiaModule.
- A .feature file specifying your dependent jars and module class.
- A module.properties file is required in the custom module.
- A jar containing your compiled classes with the properties file provided by UCS Director.

Before You Begin
Refer to FooModule in the sample project in the SDK.

Procedure

Step 1
Extend AbstractCloupiaModule
When extending AbstractCloupiaModule, you must register all your custom components in this class.

Step 2
Provide a .feature file.
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
Package the properties file at the root level of your module jar.
UCS Director provides you with a properties file for validation purposes. The SDK example provides you with a build file that handles the packaging process.

Note: The module.properties file is described in Publish the Module, on page 8.
CHAPTER 6

Working with a Module

This chapter contains the following sections:

- About the Module and Its Components, page 15
- Object Store, page 16
- Annotations, page 18
- Lists of Values (LOVs), page 18
- Tables (Tabular Reports), page 19
- Tasks, page 19

### About the Module and Its Components

A module can include the following components:

- **Task** — A Workflow Task that can be used while defining a Workflow.

- **Reports** — Reports that appear in the UCS Director UI. Reports may or may not contain action buttons.

- **Wizard** — A user interface component that serves to collect inputs from the user to perform a certain action or actions.

  > **Note**
  > Wizards are not supported in this release, and are not discussed in this documentation.

- **Trigger** — A condition that, once satisfied, can be associated with some action. Examples: shutdown VM, start VM, and so on.

  > **Note**
  > Triggers are not supported in this release, and are not discussed in this documentation.

Each of the major components listed above can make use of one or more of the following sub-components:

- Object Store
The Object Store provides simple APIs for database persistence. A module that needs to persist objects into the database typically uses the Object Store APIs to perform all the CRUD (Create, Read, Update, and Delete) operations.

Cisco UCS Director uses MySQL as its database. The platform runtime makes use of the Java Data Object (JDO) library provided by DataNucleus to abstract all the SQL operations through an Object Query representation. This simplifies and speeds up the development with respect to data persistence. The Object Store documentation include sections that show how CRUD operations are realized using JDO.

### Marking a Class for Persistence

A POJO class that needs to be persisted in the database has to be defined and marked with suitable JDO annotations. The class shown below is marked for JDO persistence.

In this class, note that:

- `foo_netapp_filer` is attached on top of the class declaration.
- The `table` attribute specifies the name of the table to be used.
- `foo` is the name of the module.
- `@Persistent` is attached to the field that needs persistence.

```java
package com.cloupia.lib.cIaaS.netapp.model;

@PersistenceCapable(detachable = "true", table = "foo_netapp_filer")
public class NetAppFiler {
  @Persistent
  private String filerName;

  @Persistent
  private String accountName;

  @Persistent
  private String dcName;
}
```

---

**Note**

This documentation uses the acronym POJO (Plain Old Java Object) to refer to a java class that does not extend any other class or implement any interfaces.

---

### Object Store

Cisco UCS Director uses MySQL as its database. The platform runtime makes use of the Java Data Object (JDO) library provided by DataNucleus to abstract all the SQL operations through an Object Query representation. This simplifies and speeds up the development with respect to data persistence. The Object Store documentation include sections that show how CRUD operations are realized using JDO.
The above class has two annotations: `@PersistenceCapable` and `@Persistent`. These are defined in the JDO, and the Cisco UCS Director Platform runtime expects all persistent classes to be marked with these two annotations. Cisco UCS Director uses a flat schema, so creating a nested schema, though possible and allowed in JDO, is not recommended in a Module.

**What To Do Next**

The persistence class is now ready for CRUD operations against the database.

**Publishing the Persistence Class**

A class that is marked with suitable JDO annotation has to be published so that the Platform Runtime can pick up the class.

To publish a persistence class:

**Procedure**

1. **Step 1** Create a file with the name `jdo.files` in the same directory (package) as that of the persistence class.
2. **Step 2** Add the name of the class to the file as follows:

   ```
   Example:
   Linux# cat jdo.files
   // Copyright (C) 2010 Cisco Inc. All rights reserved.
   //
   // Note: all blank lines and lines that start with // are ignored
   //
   // Each package that has Persistable Objects shall have a file called jdo.files
   // Each line here indicates one class that represents a persistable object.
   //
   // Any line that starts with a + means package name is relative to current package
   // If a line starts without +, then it must be complete fully qualified java class name
   // (for example: com.cloupia.lib.xyz.MyClass)
   +NetAppFiler
   Linux#
   ```

**Performing CRUD Operations on the Persistence Class**

When a persistence class is ready for CRUD operations against the database, you can perform the different operations available, as shown in the following examples.

**Create a New Instance of the Object**

```java
NetAppFiler filer = new NetAppFiler();
NetAppFiler filer.setAccountName("netapp-account");
filer.setDcName("Default Datacenter");
filer.setFilerName("filer0");
filer.setIpAddress("192.168.0.1");
ObjStore store = ObjStoreHelper.getStore(NetAppFiler.class);
store.insert(filer);
```
Modify a Single Instance of the Object

```java
ObjStore store = ObjStoreHelper.getStore(NetAppFiler.class);
String query = "filerName == 'filer0'";
// Use Java field names as parameter,
store.modifySingleObject(query, filer);
```

Querying All the Instances from the Database

```java
ObjStore store = ObjStoreHelper.getStore(NetAppFiler.class);
List filerList = store.queryAll();
```

Querying the Instances with a Filer Query

```java
ObjStore store = ObjStoreHelper.getStore(NetAppFiler.class);
String query = "dcName == 'Default Datacenter'";
List filerList = store.query(query);
```

Annotations

Annotations are one of the most crucial parts of Module development. Most of the artifacts are driven by annotations. This makes the development effort all the more easy and convenient.

Annotations are used for persistence, report generation, wizard generation, and tasks.

Persistence Annotations

See Marking a Class for Persistence, on page 16, for information about the annotations that are used for persistence.

Task Annotation

When a task is included in a Workflow, the user is prompted for certain inputs. The user is prompted for an input when a field of the class representing the task is marked with an annotation. TheFormField annotation determines what type of UI input field to show to the user: a text field, or a dropdown list, or a checkbox, etc. For more information, see Tasks, on page 19.

Lists of Values (LOVs)

Lists represent the drop-down LOVs (Lists of Values) that are displayed to the user to facilitate getting the correct inputs for a task. You can reuse an existing list or create your own list to show in the Task UI. Cisco UCS Director defines over 50 prebuilt List providers that the modules can readily use to prompt input from the user. For more information, see List Providers.

For an example that shows how to use one of the list providers, see Defining Your Own List Provider, on page 18, and Tasks, on page 19.

Defining Your Own List Provider

You can define your own list provider and ask the Platform Runtime to register it with the system.
A list provider class implements the `LOVProviderIf` interface and provides implementation for the single method `getLOVs()`. See the following example:

```java
class MyListProvider implements LOVProviderIf {

    /**
     * Returns array of FormLOVPair objects. This array is what is shown
     * in a dropdown list.
     * A FormLOVPair object has a name and a label. While the label is shown
     * to the user, the name will be used for uniqueness
     */
    @Override
    public FormLOVPair[] getLOVs(WizardSession session) {
        // Simple case showing hard-coded list values
        FormLOVPair http = new FormLOVPair("http", "HTTP");
        // http is the name, HTTP is the value
        FormLOVPair https = new FormLOVPair("https", "HTTPS");
        FormLOVPair[] pairs = new FormLOVPair[2];
        pairs[0] = http;
        pairs[1] = https;
        return pairs;
    }
}
```

### Tables (Tabular Reports)

Use the existing tabular reports available for user selection. See [Existing List of Value Tables](#), on page 67.

### Tasks

Workflow Tasks provide the necessary artifacts to contribute to the Task library maintained by Cisco UCS Director. The task can be used in a Workflow definition.

At a minimum, a task should have the following classes:

- A class that implements the `TaskConfigIf` interface.
- A class that extends and implements methods in the `AbstractTask` class.

#### TaskConfigIf

A class that implements this interface becomes a Task’s input. That is, a task that wants to accept inputs for its execution shall depend on a class that implements `TaskConfigIf`. The class that implements this interface should also contain all the input field definitions appropriately annotated for prompting the user. The class should also have JDO annotations to enable the Platform runtime to persist this object in the database.

A sample Config class is shown in the sample code.

#### AbstractTask

A task implementation must extend the `AbstractTask` abstract class and should provide implementation for all the abstract methods. This is the main class where all the business logic pertaining to the task goes. The most important method in this class, where the business logic implementation will be scripted, is `executeCustomAction()`. The rest of the methods provide sufficient context to the Platform runtime to enable
the task to appear in the Orchestration designer tree and to enable the task to be dragged and dropped in a Workflow.
About Workflow Tasks

A task is composed of implementations of TaskConfigIf and AbstractTask. The TaskConfigIf is used to specify all the data that you need, and the AbstractTask is where you execute all your logic.

Developing a TaskConfigIf

To develop a task, you must first implement TaskConfigIf. During the process of setting up the task configuration interface, you must determine what data is required to perform your task.

In the following example, EnableSNMPConfig exposes details of the process of developing a TaskConfigIf. The Enable SNMP task is designed to enable SNMP on a Cisco Nexus device.

To proceed, you must have the IP address of the Nexus device, the login, and the password.

You see the annotation at the beginning of EnableSNMPConfig.

```java
@PersistenceCapable(detachable= "true", table = "foo_enable_snmp_config")
public class EnableSNMPConfig implements TaskConfigIf {
```
You must provide a `PersistenceCapable` annotation with a table name that is prefixed with your module ID. You must follow this convention; because Cisco UCS Director prevents a task from being registered if you try to use a table name that is not prefixed with your module ID.

Next, see the following fields:

- **handler name**
- **configEntryId**
- **actionId**

```java
public static final String HANDLER_NAME = "Enable SNMP for Nexus";

//configEntryId and actionId are mandatory fields
@Persistent
private long configEntryId
@Persistent
private long actionId
```

The handler name is the name of the task. The name should be a unique string; you will create problems if you use the same handler name in multiple tasks.

Each task must have a `configEntryId` and `actionId`, exactly as shown above. You must have corresponding getter and setters for these two fields. These two fields are absolutely mandatory; you must have these fields in your config object.

Next, you see the data actually needed to perform the task:

```java
//This is the ip address for the Nexus device on which you want to enable SNMP.
@FormField(label = "Host IP Address", help = "Host AP Address", mandatory = true,
     type = FormFieldDefinition.FIELD_TYPE_EMBEDDED_LOV,
     lovProvider = ModuleConstants.NEXUS_DEVICES_LOV_PROVIDER)
@UserInfoField(type = ModuleConstants.NEXUS_DEVICE_LIST)
@Persistent
private String ipAddress = "";

@FormField(label = "Login", help = "Login", mandatory = true
@Persistent
private String login;

@FormField(label = "Password", help = "Password", mandatory = true
@Persistent
private String password;
```

As you review the code sample above, note that the developer needs the following:

- The IP address of the device.
  In this example, an LOV is used to get this IP address. See Annotations, on page 18 for more information about annotations and LOVs.

- The login and password, which the user must enter.
  To obtain these, use the form field annotations to mark these fields as data that will be provided by the user.

- Getters and setters for each of these fields.

Once the config object is completed, you must mark it for Java Data Object (JDO) enhancement.

**Before You Begin**

You must have the UCS Director Open Automation software development kit (SDK).
Procedure

Step 1
Include a jdo.files file in the same package as your config objects. See the jdo.files and packaging in the SDK example. Note that the jdo.files must be named exactly in this way.

Step 2
In the jdo.files, specify all the classes that need to go through JDO enhancement. The build script supplied with the SDK will complete JDO enhancement for you if you have executed this step properly.

What to Do Next
The handler object is where you actually execute your custom code. A handler object must implement AbstractTask. The executeCustomAction method enables you to retrieve the corresponding config object that you developed previously to execute your code.

Developing an Abstract Topic

When your config object is ready, you must extend AbstractTask to actually use the new config object. This example shows the EnableSNMPTask.

At this point, you should look at this method: executeCustomAction.

```java
public void executeCustomAction(CustomActionTriggerContext context, CustomActionLogger
actionLogger) throws Exception
{
    long configEntryId = context.getConfigEntry().getConfigEntryID();
    //retrieving the corresponding config object for this handler
    EnableSNMPConfig config = (EnableSNMPConfig) context.loadConfigObject();
    executeCustomAction is where the custom logic takes place. When you call context.loadConfigObject(), you can cast it to the config object that you defined earlier. This process allows you to retrieve all the details that you need to perform your task. This example shows that after getting the config object, the SSH APIs are used to execute the enable SNMP commands.

    When a workflow is rolled back, a task must provide a way to undo the changes it has made. This example shows the use of a change tracker:

    //If the user decides to roll back a workflow containing this task,
    //then using the change tracker, we can take care of rolling back this task (i.e.,
    //disabling snmp)
    context.getChangeTracker().undoableResourceAdded("assetType", "idString",
    SNMP enabled", "SNMP enabled on " + config.getIpAddress(),
    new DisableSNMPNexusTask().getTaskType(), new DisableSNMPNexusConfig(config));
```

The rollback code informs the system that the undo task of Enable SNMP task is the Disable SNMP task. You provide the undo config object and its name. The rest of the arguments are about logging data, which you might or might not want to provide.

**DisableConfig** actually takes place in the **EnableConfig**. In this case, the enable config contains the device details, so when the Disable SNMP task is called, you know exactly which device to disable SNMP on.

You must also implement getTaskConfigImplementation. This example instantiates an instance of the config object in returning it:

```java
@override
public TaskConfigIf getTaskConfigImplementation() {
```
Registering Custom Workflow Inputs

You can develop your own input types in Cisco UCS Director. For more information, refer to Cisco UCS Director Orchestration Guide, Release 4.1. However, they must be prefixed with your module ID. See Developing a TaskConfigIf, on page 21, in which an additional annotation is used to specify a custom workflow input.

```java
public static final String NEXUS_DEVICE_LIST = "foo_nexus_device_list";
@UserInputField(type = ModuleConstants.NEXUS_DEVICE_LIST)
```

In this example, ModuleConstants.NEXUS_DEVICE_LIST resolves to foo_nexus_device_list.

**Before You Begin**

Develop the required TaskConfigIf and the AbstractTask components for your custom workflow.

**What to Do Next**

Register a custom workflow output. See Registering Custom Task Output, on page 24.

Registering Custom Task Output

You can enable a task to add an output.

**Before You Begin**

See the EmailDatacentersTask to see an example of how to create custom task outputs.

**Procedure**

1. **Step 1** Implement the method getTaskoutputDefinitions() in the task implementation and return the output definitions that the task is supposed to return.

   ```java
   @Override
   public TaskOutputDefinition[] getTaskOutputDefinitions() {
       TaskOutputDefinition[] ops = new TaskOutputDefinition[1];
       ops[0] = FooModule.OP_TEMP_EMAIL_ADDRESS;
       return ops;
   }
   ```

2. **Step 2** Set the output from the task implementation.

   ```java
   @Override
   public void executeCustomAction(CustomActionTriggerContext context,
                                   CustomerActionLogger action.Logger) throws Exception {
       long configEntryId = context.getConfigEntry().getConfigEntryId();
       //retrieving the corresponding config object for this handler
       EmailDatacentersConfig config = (EmailDatacentersConfig context.loadConfigObject());
   }
   ```
if (config == null) {
    throw new Exception("No email configuration found for custom Action"
        + context.getAction().getName
        + "entryId" + configEntryId);
}

try {
    context.saveOutputValue(OutPutConstants.OUTPUT_TEMP_EMAIL_ADDRESS, toAddresses);
}

## Consuming Custom Output as Input in Other Tasks

This section describes how output can be used as input in another task. This section uses some aspects of the example in the previous section. The output definition is defined as follows:

```java
@Override
public TaskOutputDefinition[] getTaskOutputDefinitions() {
    TaskOutputDefinition[] ops = new TaskOutput Definitions[1];
    //NOTE: If you want to use the output of this task as input to another task. Then the second argument
    //of the output definition MUST MATCH the type of UserInputField in the config of the task that will
    //be receiving this output. Take a look at the HelloWorldConfig as an example.
    ops[0] = new TaskOutputDefinition(
        FooConstants.EMAIL_TASK_OUTPUT_NAME,
        FooConstants.FOO_HELLO_WORLD_NAME,
        "EMAIL IDs");
    return ops;
}
```

The example defines an output with the FooConstants.EMAIL_TASK_OUTPUT_NAME name, and with the FooConstants.FOO_HELLO_WORLD_NAME type. To configure another task that can consume the output as input, you must make the types match.

So, in the new task that consumes FooConstants.FOO HELLO_WORLD_NAME as input, you must enter the following in the configuration object:

```java
//This field is supposed to consume output from the EmailDatacentersTask.
//You'll see the type in user input field below matches the output type
//in EmailDatacentersTasks's output definition.
@FormField(label = "name", help = "Name passed in from a previous task", mandatory = true)
@UserInputField(type = FooConstants.FOO_HELLO_WORLD_NAME)
@Persistent
private String login;
```

The type in the UserInputField annotation matches the type that is registered in the output definition. With that match in place, when you drag and drop the new task in the Cisco UCS Director Workflow Designer, you can map the output from one task as input to the other task while you are developing the workflow.

## Consuming Output from Existing Tasks as Input

This section shows how to consume output from built-in workflow tasks as input to your custom task. This process is similar to setting up custom outputs to be consumed as input in one important way: the configuration object of your task must have a field whose type is exactly the same as the type of the output that you want.
Procedure

**Step 1** Choose Policies > Orchestration > Workflows, and then click Task Library. Tip Press Cntl–Find to locate tasks in the very long list that appears. For example, entering Group takes you directly to User and Group Tasks.

**Step 2** Find the task that you want to add, and then choose it to see the information displayed under the heading: User and Group Tasks: Add Group. Tip Press Cntl–Find to locate tasks in the very long list that appears. For example, entering Group takes you directly to User and Group Tasks. The crucial type data is provided in the Outputs table, the last table provided under the heading.

<table>
<thead>
<tr>
<th>Table 2: Add Group - Outputs Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>OUTPUT_GROUP_NAME</td>
</tr>
<tr>
<td>OUTPUT_GROUP_ID</td>
</tr>
</tbody>
</table>

**Step 3** Pick the appropriate Type value from the Outputs table. The goal is to obtain the Type value that will be matched to the Task. In the example, the task consumes the group ID, so you know that the Type is gen_text_input.

**Step 4** Specify the Type value in the UserInputField.

Example:

```java
@FormField(label = "Name", help = "Name passed in from previous task", mandatory = true)
@UserInputField(type="gen_text_input")
@Persistent
private String name;
```

You could also use @UserInputField(type=WorkflowInputFieldTypeDeclaration.GENERIC_TEXT). This is equivalent to using @UserInputField(type="gen_text_input"). You may find it easier to use type = WorkflowInputFieldTypeDeclaration.GENERIC_TEXT which uses the constants defined in the SDK.

**Note** The last step is to configure the mapping properly when you are developing your workflow.

**Step 5** Configure the mapping as you develop your workflow, using the User Input Mapping to Task Input Attributes window as you add an action to the workflow, or edit related information in the workflow.

Verifying the Custom Task Is In Place

Assuming that your module is working properly, you can verify that the custom task is in place by opening the Cisco UCS Director Task Library and verifying that the task appears in it.
Procedure

Step 1  In Cisco UCS Director, choose Policies > Orchestration, and then choose the Workflows tab. The Workflows tab displays a table that lists all available workflows.

Step 2  In the Workflows tree directory, navigate to a workflow in which the task appears, and then choose that workflow row. To facilitate navigation, use the Search option in the upper right-hand corner, above the table, to navigate to the workflow.

Additional workflow-related controls appear above the workflows table.

Step 3  With workflow selected, click Workflow Designer.

The Workflow Designer screen opens, displaying an Available Tasks list and the Workflow Design graphic view.

Step 4  Verify that the task of interest appears in the list of available tasks and in the graphic representation of the tasks in the workflow.

Debugging Tasks

When you are debugging, you can trace problems through inframgr (Infra Manager) logs. You can get the inframgr logs through either the Cisco (CUIC) Shell menu or the Cisco UCS Director GUI.

To use the shell admin approach:

- Open the Cisco (CUIC) Shell menu.
- Choose 18, Tail Inframgr logs.

To use the Cisco UCS Director GUI:

- Log in to the Cisco UCS Director GUI.
- Navigate to Administration > Support Information.
- Click System Information and Logs Link Page.
- In the Cisco UCS Director Product System Information form, from the drop-down menu, choose Infra Manager and click Show Log.

Before You Begin

You will need shell admin access in order to use the debugging functions that are available in the Cisco (CUIC) Shell menu (also known as shell admin). Refer to Deploying the Module on Cisco UCS Director, on page 10, for more details about access to and use of the Cisco (CUIC) Shell menu.
Developing Your Own Inventory Collector

This chapter contains the following sections:

- About the Inventory Collector, page 29
- Guidelines for Developing a Module, page 30
- Connecting to the Device (NodeConnectorIf), page 31
- Binding Collected or Parsed Data to POJO (ItemDataObjectBinderIf), page 31
- Using Generic Infra Account Report to Simplify Account Management, page 32
- Registering Inventory Collection Components and Account Management Reports, page 32
- Test Connection Handler (Optional), page 32
- Converged Stack Builder (Optional), page 32
- Stack View Provider (Optional), page 33

About the Inventory Collector

You can introduce support for new devices by implementing your own inventory collector using the collector framework. When adding support for new devices, you must implement your inventory collector to handle collection and persistence of data in the database.

Accompanying the inventory collector framework is a set of reports for displaying the data. For more information, see About Reports, on page 41.

**CollectorFactory**

The main class is the CollectorFactory (com.cloupia.service.cIM.inframgr.collector.controller.CollectorFactory). When you create a new CollectorFactory, you are introducing a new type of account. All supported devices correspond to a built-in account type. You must provide a unique integer so the system can uniquely identify this new account. You should start with a large number such as 1000, to prevent potential conflicts with built-in account types.

You then need to specify the account category: compute, storage or network. The system handles collectors differently depending on the category, so make sure that you select the proper one.
You can optionally provide a connection tester, which will be called when you try to test a connection in the User Interface (UI). A base implementation is already done (GenericInfraAccountConnectionTestHandler). To use the base implementation, fill in the logic to perform a connection to the device. You can also provide a converged stack builder, which displays the details of the device in the Converged tab of the UI. For more information, see Test Connection Handler (Optional), on page 32.

The most important thing to provide in your CollectorFactory is the InventoryCollector. To set up your InventoryCollector, specify the account for the collector, and what items to collect. You need to provide implementations of NodeID and ItemIf. A GenericNodeID can locate the details of the account which you can use. When you implement ItemIf, you are describing the type of data that you would like to collect.

InventoryCollector handles inventory collection. The most important classes are CollectorFactory and InventoryCollector.

**InventoryCollector**

Inventory collection includes four phases:

1. Connection
2. Parsing
3. Binding
4. Persistence

In phase 1, you need to establish a connection to the device. This phase is where you establish a Secure Shell (SSH) session or retrieve the credentials to access a REST API.

In phase 2, data collection begins and the collected data is parsed. This second phase is OPTIONAL. It is only necessary if the data you collected is unstructured. For example, if you are collecting CLI output, it might not be in a format that is ready for usage yet. In this parsing phase, you should convert this output into a more suitable format for development, such as XML or JSON. Once that data is in a usable format, it is passed along into the binding phase. Phase 2 is typically optional because most devices today provide developer friendly APIs to communicate with devices.

In phase 3, you convert whatever data you received in the parsing phase so it is ready to be persisted in the database. In this phase, you convert that data into a JDO POJO so it can be persisted in the final phase. You need to complete Phase 1 and Phase 3 to provide your own implementations.

In phase 4, the output from the device has been converted into a JDO POJO that is ready to be persisted into the database. This phase is OPTIONAL. A default implementation can do this if you are returning JDO POJOs. For phase 4, you can usually use the base implementation to handle persistence. However, you might need to write your own code if you are trying to persist a complicated database model.

Your implementation of Inventory Collector needs to provide some means of handling these four phases.

---

**Guidelines for Developing a Module**

When you are developing a new module to support new devices, you should develop modules for a device family. Which means that you should have only one module to support devices that are similar. It also means that you should not develop a module that supports both a network switch and a storage controller; instead, you should split them into two modules. Ideally, a module should support only devices within the same category, so a module would handle only compute devices or network devices, or storage devices. In addition to this distinction, the devices supported by the same module should be relatively similar too. The same device
might come in different models that are meant for distinct purposes, and it might be appropriate to use different modules to support them.

## Connecting to the Device (NodeConnectorIf)

Your implementation handling the connection phase will implement the `NodeConnectorIf` interface. A `NodeConnectorIf` is used for each item in the following flow:

- `connect()`, for each `ItemIf` you specified in the `CollectorFactory`
- `getItem()`
- `disconnect()`

The `connect` method provides you with a `ConnectorCredential` and `ConnectionProperties`. The `ConnectorCredential` has a method `toInfraAccount` which provides you the details of the device you are trying to connect to, such as, IP address, the login, and password credentials. You must use this data to execute your connection logic, and return a new instance of `ConnectionStatus`, the `ConnectionStatus` object should be set connected to true or false.

The `getItem` method provides you with a `NodeID` (the same one you provided when you created your `InventoryCollector`) and the `ItemIf` that represents the data that you are currently trying to collect. You can use the `NodeID` to retrieve device details if you need them. The `ItemIf` should tell you what you are trying to collect. You execute your collection logic in this method and then return it wrapped in an `ItemResponse`. The `ItemResponse` should contain the collected data, with a reference to the `NodeID` and `ItemIf`, so it can be used in later phases.

The `disconnect` method is used to handle any clean up after collection is completed.

## Binding Collected or Parsed Data to POJO (ItemDataObjectBinderIf)

In the binding phase, you must convert whatever data you have collected or parsed into a POJO so it can be ready for persistence. You do this process with the `bind` method. Binded objects should implement the `InventoryDBItemIf` interface. The purpose of this interface is to ensure that you include an `accountName` field in your `InventoryDBItem`. This process will in turn allow you to filter on the account name in your reports, resource limit computations, and so on.


## Using Generic Infra Account Report to Simplify Account Management

Now that you have an inventory collector task that is ready to use, you need to tell the system to use the collector. You need some way to add an account, which will trigger the system to locate your collector and begin collection.
The easiest way to accomplish this task is to extend `GenericInfraAccountReport`, which provides a base implementation that helps you to add new accounts into the system and allows you to edit and delete accounts. You can then specify the unique account type and account category. The report is available in this location Physical > Compute/Network/Storage, depending on the type you specify.

You should also supply the other reports that you want to display in the same location. When you drill down on a report row supplied by `GenericInfraAccountReport`, it displays the other reports that you are providing. For more information about implementing your own reports, see About Reports, on page 41.

This report has several basic account management actions enabled, so you do not have to create them yourself. When you drill down on a specific account, you should see the reports that you specified as drill-down reports.

**Before You Begin**

You have an inventory collector task ready for use.

## Registering Inventory Collection Components and Account Management Reports

By now you should have implemented your own inventory collection components and account management reports. The final step is to register all these components in your `AbstractCloupiaModule`. Specifically, you must implement the `getReports()` and `getCollectors()` methods. Instantiate and return new instances of the collectors and reports and they will be registered into the system.

## Test Connection Handler (Optional)

When you develop a new Compute or Storage collector, all the account details are stored in the Physical Accounts report under Administration > Physical Accounts. When a user selects a row, the Test Connection option appears. If you provide a test connection handler in your CollectorFactory, when the user executes this test action, this handler is called to retrieve the connection status.

Cisco UCS Director provides an implementation of this test connection handler, `com.cloupia.service.cIM.inframgr.GenericInfraAccountConnectionTestHandler`. To set up this connection status option, you need to implement one function, `testConnectionTo`. You can use the `InfraAccount` object to retrieve the device details that you will be testing. The `StringBuffer` enables you to display additional information to the user. Test Connection returns `true` if the connection succeeds; it returns `false` if it fails.

## Converged Stack Builder (Optional)

In the Converged tab of the UI, Cisco UCS Director displays the converged stack of devices for a data center. When developing a new CollectorFactory, if you want to display your device in the Converged UI, you must supply your own `ConvergedStackComponentBuilderIf`, a device-icon mapping file, and the icons you would like to show.

---

**Note**

The data center type must be Generic. Devices that are introduced through Open Automation are visible only in the Converged UI if the data center type is specified as Generic. If you specify VSPEx, FlexPod, or anything else, the device does not appear because the device is not considered as part of those stacks.
Before You Begin

Files in the sample code, including:

- device_icon_mapping.xml
- dcom.cloupia.feature.foo.inventory.DummyConvergedStackBuilder
- the resources folder containing all the images

Procedure

**Step 1** Provide an implementation of ConvergedStackComponentBuilderIf.
Extend the abstract implementation: com.cloupia.service.cIM.inframgr.reports.contextresolve.AbstractConvergedStackComponentBuilder.

**Step 2** Supply a device icon mapping file.
This XML file is used to map the data supplied by your ConvergedStackComponentBuilderIf to the actual images to be used in the UI. This XML file must be named device_icon_mapping.xml and it must be packaged inside your jar.

**Important** For each entry in the XML file, the DeviceType must match the Model in the ComponentBuilder and the Vendor must match the Vendor in the ComponentBuilder. The framework uses the vendor and model to uniquely identify a device and to determine which icon to use. Also, in the XML file, the IconURL value should always start with /app/uploads/openauto. All of your images will be dumped into this location.

**Step 3** Make sure that your images are packaged in your module.zip in a folder called resources. The framework copies all your images into resources and places them in an uploads folder.

Stack View Provider (Optional)

If you want to display data that specific to your device in the Stack View, which is located under Virtual > Compute > VMs > Stack View, you can provide your own implementation of com.cloupia.model.cIM.stackView.StackViewItemProviderIf in your CollectorFactory.

This interface has several implementations that are specific to the device category:

- If you are working on a storage category device, use com.cloupia.service.cIM.inframgr.stackView.AbstractOAStorageStackViewProvider.
- If you are working on a network category device, use com.cloupia.service.cIM.inframgr.stackView.AbstractOANetworkStackViewProvider.
- If you are working on a compute category device, use com.cloupia.service.cIM.inframgr.stackView.AbstractOAComputeStackViewProvider.

For each implementation, you must determine if your device applies to the Virtual Machine (VM) stack and what data you would like to display in the UI.

The VM stack currently works if you use vCenter for provisioning; it might not work if you use HyperV.
Developing Your Own Inventory Collector

Stack View Provider (Optional)
Developing a New Menu

This chapter contains the following sections:

- About New Menus, page 35
- Defining a Menu Item, page 35
- Registering a Menu Item, page 37
- Defining Menu Navigation, page 38
- Registering Report Contexts, page 39

About New Menus

Creating a new menu item involves a series of associated tasks:

- Define your menus.
- Register new menus.
- Register new report contexts.
- As necessary, write a left hand navigation tree provider.
- Write your reports and make sure they point to your new menu location with the proper context map rule.

Defining a Menu Item

When using Open Automation, the only way to add new menus is by providing a file called menu.xml. The code examples below provide guidance regarding the task of defining a menu item.

The following code samples are included in the Open Automation SDK samples. The XML samples show two options for introducing menu items:

- How to insert new menu items into existing folders (for example, the Virtual folder)
- How to add entirely new menu items into the UI
Before You Begin

Every menu node has a unique integer ID associated with it. When introducing new menu items, be aware that the following menu IDs are reserved as indicated in the table.

<table>
<thead>
<tr>
<th>Menu</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual</td>
<td>1000</td>
</tr>
<tr>
<td>Physical</td>
<td>1001</td>
</tr>
<tr>
<td>Organizations</td>
<td>1002</td>
</tr>
<tr>
<td>Policies</td>
<td>1003</td>
</tr>
<tr>
<td>Virtual/Hypervisor Policies</td>
<td>1007</td>
</tr>
<tr>
<td>Physical Infrastructure Policies</td>
<td>1008</td>
</tr>
<tr>
<td>Administration</td>
<td>1004</td>
</tr>
</tbody>
</table>

Listed below are the components of a Menu Item:

- **label** - This is the label displayed in the UI for the menu.
- **path** - This is the value used in the URL when navigating to the menu.

**Important** You must include a backslash at the end of this string; for example: dummy_menu_1/ or dummy_menu_2/

- **op** - This is the permission the user must have in order to access this menu; no_check means everyone can access it.
- **url** - This should always be "modules/GenericModules.swf". This field should only be populated if it's a menu item NOT category.
- **leftNavType** - Valid values are either "none" or "backend_provided". We'll discuss what theses values mean in details in a later section. This field should only be populated if it's a menu item NOT category.
- **children** - If the menu has more menus underneath it, you add them here. The best practice is not to have more than three levels in a menu.

Sample values for the menu item components listed above are provided in the code examples below.

**Procedure**

**Step 1** Option 1: Add a new menu item underneath an existing folder; in this case, the one called Virtual.

When adding a menu item into an existing menu category, you first have to locate the menuid of the category to which you want to add the item. In the example, we add the new menu item under "Virtual", which has the menuid of 1000. Take note of the parent menu item with just the menuid filled in: this is all you need in order
to signal that you are placing your menu item into an existing category. The new menu item is placed into the children field.

Example:

```xml
<menu>
<!-- this shows you how to add a new menu item underneath virtual -->
<menuitem>
  <menuid>1000</menuid>
  <children>
    <menuitem>
      <menuid>12000</menuid>
      <label>Dummy Menu 1</label>
      <path>dummy_menu_1/</path>
      <op>no_check</op>
      <url>modules/GenericModule.swf</url>
      <leftNavType>backend_provided</leftNavType>
    </menuitem>
  </children>
</menuitem>
</menu>
```

**Step 2**

Option 2: Add an entirely new menu item into the UI.

If you are defining an entirely new menu item, provide all the details as shown in the example. First provide all the details for the menu category, then add all the child menu items underneath it. The example here shows a menu two levels deep, but in theory you can go as deep as you want. The best practice is to create menus no more than three levels deep.

Example:

```xml
<!-- entirely new menu -->
<menu>
  <menuitem>
    <menuid>11000</menuid>
    <label>Whitney Category</label>
    <path>whitney/</path>
    <op>no_check</op>
    <children>
      <menuitem>
        <menuid>11001</menuid>
        <label>Whitney Menu 1</label>
        <path>whitney_menu_1/</path>
        <op>no_check</op>
        <url>modules/GenericModule.swf</url>
        <leftNavType>backend_provided</leftNavType>
      </menuitem>
    </children>
  </menuitem>
</menu>
```

**What to Do Next**

Register the menus.

**Registering a Menu Item**

For Open Automation, menu registration is handled automatically. As a Developer, you only need to name your menu xml as "menu.xml", then package it as part of your module. "menu.xml" must sit in the top level of your module jar.
Defining Menu Navigation

Cisco UCS Director uses menu navigation to determine what reports and forms to display in the UI. For more information on the subject of report locations, refer to Specifying the Report Location, on page 44.

The leftNavType field specifies the type of navigation to be used in your menu item. The value "none" means that:

- No navigation is required.
- The context map rule associated with the menu item will use type = 10, name = "global_admin".

(Important!) When the leftNavType is set to "none, the type value and name value for the context map rule associated with the menu item will come in handy when you need to register your reports to this menu location!

If the leftNavType is "backend_provided", then you must provide an implementation of com.cloupia.model.cIM.AbstractTreeNodesProvider if that populates the left hand navigation tree. Each node of this navigation tree will need to provide the following elements:

- a label
- the path to an icon to show in the UI (optional)
- the context type (for more details, see the section about registering report contexts)
- the context ID (this will become the report context ID that you may use when generating tables)

The navigation tree needs to be associated with a menu id, so when registering the tree provider, make sure to use the corresponding menu id.

Refer to code samples for more detail.

Related Topics

t_register_report_contexts.xml
Registering Report Contexts

This topic focuses on adding new report contexts. When developing new menu items, new report contexts are crucial: you must register new unique contexts, you CANNOT use existing contexts.

The Open Automation documentation about defining menu navigation briefly mentions that you need to provide a report context type when building your left hand navigation tree provider.

Report contexts are used by the system to determine which reports can be displayed at any point in the UI. For more background information, refer to the documentation on specifying report location: Specifying the Report Location, on page 44. See also the list of existing report context data in Appendix B, on page 71.

For open automation, there are APIs in place to auto-generate a new report context. Refer to com.cloupia.feature.foo.FooModule for examples on registering report contexts and menu providers.

Tip

Auto generated report contexts are not portable. This means that if you deploy your module in one instance of UCSD and the same module in another instance of UCSD, the auto-generated report context you get in each instance may have different values. Thus, any code you write that uses those duplicate values will not necessarily work! To avoid such problems, use the ReportContextRegistry to register report contexts and retrieve them.

Use com.cloupia.model.cIM.ReportContextRegistry.register(String name, String label), and take a look at the javadocs and sample code for more detail.

Refer to code samples and the Specifying Report Location document to see how these report contexts ultimately end up being used.

Before You Begin

Open Automation developers who need to register report contexts should first talk to a UCSD lead. The UCSD lead can provide you with a block of integers reserved exclusively for your use. This will guarantee that any report contexts you define are unique. When you have your block, you can use ReportContextRegistry.register(int type, String name, String label) to register the new context.
Developing Your Own Reports

This chapter contains the following sections:

- About Reports, page 41
- Developing Reports Using POJO and Annotations, page 42
- Using TabularReportGeneratorIf, page 43
- Drillable Reports, page 43
- Registering Reports, page 44
- Specifying the Report Location, page 44
- Developing Bar Chart Reports, page 45
- Developing Line Chart Reports, page 46
- Developing Pie Chart Reports, page 47
- Developing Heat Map Reports, page 48
- Developing Summary Reports, page 49
- Developing Form Reports, page 51

About Reports

You can develop your own reports in two ways. The simplest way is to use the Plan Old Java Object (POJO)-and-Annotation approach. The more advanced approach is to implement the TabularReportGeneratorIf interface programmatically.

You can develop POJO-based reports with the following classes:

- CloupiaEasyReportWithActions
- CloupiaEasyDrillableReport

You can develop programmatic reports with the following classes:

- CloupiaReportWithActions
Developing Reports Using POJO and Annotations

The simplest way to develop a report is to use the JDO POJOs that you developed for persistence and add some annotations. The report will display in the user interface (UI). Typically, whatever data you persist in the database also happens to be the same data that you want to show in the UI.

Before You Begin

See com.cloupia.feature.foo.inventory.model.DummyInterface in the sample code for an example of a report data source POJO.

Procedure

Step 1

Your data source POJO must implement the com.cloupia.service.cIM.inframgr.reports.simplified.ReportableIf interface.

The ReportableIf interface requires that you use the getInstanceQuery method which should return a predicate and is used by the framework to filter out any instances of the POJO that you do not want to display in the report.

Step 2

For each field in the POJO you want to show in the report, use the @ReportField annotation to mark it as a field to include in the report. This annotation supports only primitive types such as int, String, and so on.

Example:

```java
@ReportField(label = "Name")
@Persistent
private String name;

@ReportField(label = "Size")
@Persistent
private String size;
```

See com.cloupia.feature.foo.inventory.model.DummyInterface in the sample code to see how a report data source POJO should look. Once you have your POJO ready, you can refer to it as the data source.

Step 3


You must provide the report name (some string to uniquely identify this report), the label of this report (to be displayed to the user), and the data source (the POJO that you just created).
For example, see com.cloupia.feature.fooreports.DummyInterfacesReport.

## Using TabularReportGeneratorIf

If you need more granular control over how you display the data in your report, you implement this interface. Use the ReportContext object, and specifically the ID of the report context, to determine what data to retrieve and how to filter it.

**Tip**

To ensure that you have access to the report context ID and related information that indicates how to use it, log the report context. Use the logger and print the report context ID, then take a look at the logs to see that context ID values.

To create an instance of TabularReportInternalModel that contains all the data that you want to show, perform the following steps:

### Before You Begin


### Procedure

#### Step 1
Create the TabularReportInternalModel. Populate the header, and then add text values for each column to fill out a row.

#### Step 2

- Specify the implementation of the data source.
- Make sure that isEasyReport() returns false.

### What to Do Next

When you complete your implementation, you can use it as a data source for a report.

## Drillable Reports

You can develop reports that are nested within other reports.

To get such reports, do the following:

- If your report's data source was implemented through the POJO and Annotation approach, your report should extend com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaEasyDrillableReport.

- If your report's data source was implemented using TabularReportGeneratorIf, your report should extend com.cloupia.service.cIM.inframgr.reports.simplified.DrillableReportWithActions.
Both classes require that you provide instances of the reports that are displayed when the user drills down on the base report.

**Important**

Each time that getDrillDownReports() is called, it should return the same instances. Do not return new instances each time that you use this method. You should initialize the array of reports and declare them as member variables, as in com.cloupia.feature.foo.reports.DummyAccountMgmtReport.

---

**Registering Reports**

The final step in developing your own Inventory Collector is to register all the components that you have developed in your AbstractCloupiaModule.

Specifically you must implement the getCollectors() and getReports(). If you instantiate and return new instances of the reports and collectors and reports, they will be registered into the system.

**Before You Begin**

You must have implemented your own inventory collection components and account management report.

---

**Specifying the Report Location**

To specify the exact location where your report will appear in the user interface, you must provide two pieces of information:

- The UI menu location's ID
- The Context Map Rule that corresponds to the report context of the location.

To gather these pieces of information, start by using the metadata provided by Cisco UCS Director. The metadata includes data for the report nearest to the place where you want your report to appear, and you can use this data to start constructing the report specifications that you need.

**Procedure**

**Step 1**

Enable the developer menus for your session.

a) In Cisco UCS Director, click your login name in the upper right.
b) In the User Information dialog box, click the Advanced tab.
c) Check the Enable Developer Menu (for this session) check box and close the User Information dialog box.

The Report Metadata option becomes available in the report views opened in the session.

**Step 2**

Navigate to a tabular report in the same location where you want your report to appear, then click on Report Metadata to see the Information window. See the Report Context section at the top of that window.

a) Find the integer value assigned to the uiMenuTag.

The uiMenuTag tells you what your report's getMenuID should return.
b) Find the value assigned to the type.
The type tells you the first piece of information you need to build the context map rule, which in turn tells you what your report’s `getMapRules` should return.

**Step 3** Get the second piece of information necessary to build the context map from the `reportContexts.html` file. See a copy in Appendix B, on page 71. The `reportContexts.html` file lists every report context registered in the system. The first column provides the type of report context and the second column provides the name of the report context. Given that you have the type, you can locate the name. For example, 0 maps to "global".

When you have both pieces of information (the context name and the context type) you can build your context map rule.

**Step 4** Instantiate a Context Map Rule with details similar to those in the following code sample.

```
Example:
ContextMapRule rule = new ContextMapRule();
rule.setContextName("global");
rule.setContextType(0);
ContextMapRule[] rules = new ContextMapRule[1];
rules[0] = rule;
```

Note that this sample uses the plain constructor. Do NOT use another constructor. The plain constructor serves the purpose and explicitly sets these values.

If your report specification code has properly set these new values OR overridden the methods to return these values, you should be able to view the report in the expected location.

---

**Tip**
All the new report samples will show up under Physical -> Network -> DummyAccount tab. Find a report by drilling down in one of the rows.

---

## Developing Bar Chart Reports

Open Automation enables you to develop non-tabular reports such as Bar Charts. Developing a bar chart is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a bar chart, you must:

- Extend `CloupiaNonTabularReport`.
- Use `getReportType`, and return `REPORT_TYPE_SNAPSHOT`.
- Use `getReportHint`, and return `REPORT_HINT_BARCHART`.

### Procedure

**Step 1** Extend `CloupiaNonTabularReport` by following the example provided here:

```
Example:
public class BarChartReport extends CloupiaNonTabularReport {
```
Developing Line Chart Reports

Developing Your Own Reports

Developing Line Chart Reports

Open Automation enables you to develop non-tabular reports such as line charts. The instructions below show how to create a line chart containing two lines, and make it clear how to specify a report with more lines. The HistoricalDataSeries Class provides historical information, where DataSample array is the set of values within the given timeframe (fromTime, toTime).

Developing a line chart is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a line chart, you must:

- Extend CloupiaNonTabularReport.
- Use getReportType, and return REPORT_TYPE_HISTORICAL.

Step 2

Override getReportType() and getReportHint(). Refer to this code snippet:

```java
private static final string NAME = "foo.dummy.bar.chart.report";
private static final string LABEL = "Dummy Bar Chart";

Step 2

@Override
public int getReportType()
{
    return ReportDefinition.REPORT_TYPE_SNAPSHOT;
}

@Override
public int getReportHint()
{
    return ReportDefinition.REPORT_HINT_BARCHART;
}

Step 3

Implement your own bar chart by following the example provided in this code:

```
Example:

public class BarChartReportImpl implements SnapshotReportGeneratorIf {
    private final int NUM_BARS = 2;
    private final String BAR_1 = "bar1";
    private final String BAR_2 = "bar2";

    @Override
    public int getReportType()
    {
        return ReportDefinition.REPORT_TYPE_SNAPSHOT;
    }

    @Override
    public int getReportHint()
    {
        return ReportDefinition.REPORT_HINT_BARCHART;
    }

    @Override
    public void setReportData(ReportData data)
    {
        // Set bar chart data
    }

    public void generateReport()
    {
        // Generate bar chart
    }
}
```


Step 4

To build a bar chart and register it to a category, follow the example provided in this section of code:

```
Example:

ReportNameValuePair[] rnv1 = new ReportNameValuePair[NUM_BARS];
rnv1[0] = new ReportNameValuePair(BAR_1, 5);
rnv1[1] = new ReportNameValuePair(BAR_2, 10);

SnapshotReportCategory cat1 = new SnapshotReportCategory();
cat1.setCategoryName("cat1");
cat1.setNameValuePairs(rnv1);
```

Developing Line Chart Reports
Developing Your Own Reports

Developing Pie Chart Reports

Open Automation enables you to develop non-tabular reports such as pie charts. A single Open Automation pie chart is not generally suited to handling more than one category, so be aware that the instructions and sample code provided here are intended to create a pie chart featuring only one category. The data set generated below for the pie chart represents five slices, each slice's value is specified as \((i+1) \times 5\).

Developing a pie chart is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a pie chart, you must:

- Extend CloupiaNonTabularReport.
- Override getReportType(), and return `REPORT_TYPE_SNAPSHOT`.
- Override getReportHint(), and return `REPORT_HINT_PIECHART`.


Procedure

Step 1
Extend CloupiaNonTabularReport by following the example provided here.

Example:

```java
public class LineChartReport extends CloupiaNonTabularReport {
    private static final string NAME = "foo.dummy.line.chart.report";
    private static final string LABEL = "Dummy Line Chart";
```

Step 2
Set up the Datasets and the HistoricalDataSeries, following the code example provided here:

These instructions show how to create a line chart containing two lines, and make it clear how to specify a report with more lines. The HistoricalDataSeries Class provides historical information, where DataSample array is the set of values within the given time frame (fromTime, toTime).

Example:

```java
//specifies how to show two lines in this report
HistoricalDataSeries[] hdsList = new HistoricalDataSeries[numLines];

HistoricalDataSeries line1 = new HistoricalDataSeries();
line1.setParamLabel("param1");
line1.setPrecision(0);
DataSample[] dataset1 = createDataset1(fromTime, toTime);
line1.setValues(dataset1);

HistoricalDataSeries line2 = new HistoricalDataSeries();
line2.setParamLabel("param2");
line2.setPrecision(0);
DataSample[] dataset2 = createDataset2(fromTime, toTime);
line2.setValues(dataset2);

hdsList[0] = line1;
hdsList[1] = line2;
report.setSeries(hdsList);
```
A single Open Automation pie chart is not generally suited to handling more than one category. The instructions and sample code provided here create a pie chart featuring one category and five slices.

**Procedure**

**Step 1** To extend CloupiaNonTabularReport, follow the example provided here:

```java
public class PieChartReport extends CloupiaNonTabularReport {
    private static final string NAME = "foo.dummy.pie.chart.report";
    private static final string LABEL = "Dummy Pie Chart";
}
```

**Step 2** To create a pie chart with one category and five slices, follow the example provided in this code:

```java
ReportNameValuePair[] rnv = new ReportNameValuePair[5];
for (int i = 0; i < rnv.length; i++)
    rnv[i] = new ReportNameValuePair("category" + i, (i+1) * 5);

SnapshotReportCategory cat = new SnapshotReportCategory();
cat.setCategoryName("");
cat.setNameValuePairs(rnv);
report.setCategories(new SnapshotReportCategory[] { cat });
```


## Developing Heat Map Reports

Open Automation enables you to develop non-tabular reports such as heat maps. A heat map represents data with cells or areas in which values are represented by size and/or color. A simple heat map provides an immediate visual summary of information.

The instructions provided in this section show how to create a heat map report showing three sections, each of which is split into four equal "child" sections, where \( i \) sets the size up to 25. Developers can continue to split sections into sections by extending the approach described here.

Developing a heat map report is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a heat map, you must:

- Extend CloupiaNonTabularReport.
- Use getReportType, and return `REPORT_TYPE_HEATMAP`. 
## Developing Summary Reports

Open Automation enables you to develop your own Summary reports. The summary report is considered a non-tabular report. Although it is a summary report in function, you can determine whether or not to display this report in the summary panel.

Developing a summary report is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a summary report, you must:

- Extend CloupiaNonTabularReport.
- Override getReportType(), and return `REPORT_TYPE_SUMMARY`.
- Override getReportHint(), and return `REPORT_HINT_VERTICAL_TABLE_WITH_GRAPHS`.
- Set isManagementReport to return True.

### Developing Your Own Reports

```java
public class BarChartReport extends CloupiaNonTabularReport {
    private static final string NAME = "foo.dummy.heatmap.report";
    private static final string LABEL = "Dummy Heatmap Chart";

    // Example: To create a heatmap report with three sections, with each section split further into four sections, follow the example provided in this code:

    for (int i=0; i<3; i++) {
        String parentName = "parent" + i;
        HeatMapCell root = new HeatMapCell();
        root.setLabel(parentName);
        root.setUnusedChildSize(0.0);

        HeatMapCell[] childCells = new HeatMapCell[4];
        for (int j=0; j<4; j++) {
            String childName = parentName + "child" + j;
            HeatMapCell child = new HeatMapCell();
            child.setLabel(childName);
            child.setValue((j+1)*25); //sets color, the color used for each section is relative, there is a scale in the UI
            child.setSize(25); //sets weight
            childCells[j] = child;
        }
        root.setChildCells(childCells);
        cells.add(root);
    }
```

Before You Begin

Procedure

Step 1 To extend CloupiaNonTabularReport, follow the example provided here:

Example:
```java
public class DummySummaryReport extends CloupiaNonTabularReport {
    private static final string NAME = "foo.dummy.summary.report";
    private static final string LABEL = "Dummy Summary";
}
```

Step 2 Override getReportType() and getReportHint(), using this code snippet:

Example:
```java
@override
public int getReportType()
{
    return ReportDefinition.REPORT._TYPE_SUMMARY;
}
/**
 * @return report hint
 */
@override
public int getReportHint()
{
    return ReportDefinition.REPORT_HINT_VERTICAL_TABLE_WITH_GRAPHS;
}
```

Step 3 Define how data will be grouped together.

Example:
```java
model.addText("table one key one", "table one property one", DUMMY_TABLE_ONE);
model.addText("table one key two", "table one property two", DUMMY_TABLE_ONE);
model.addText("table two key one", "table two property one", DUMMY_TABLE_TWO);
model.addText("table two key two", "table two property two", DUMMY_TABLE_TWO);
```

Step 4 Optional: To display a Graph or Chart in a summary panel, follow the example code provided here. Use this code in the summary chart report if you want the chart to appear in the summary panel; the default is NOT to display the report in this panel. Refer to the Bar Chart topic for more detail.

Example:
```java
//NOTE: If you want this chart to show up in a summary report, you need
//to make sure that this is set to true; by default it is false.
@override
public boolan showInSummary()
{
    return true;
}
```

Developing Form Reports

Developing Form Reports

You can utilize the Open Automation form framework to build a form that occupies the space of a report. Such form reports, which consume the space of an entire tab in the UI (normally reserved for reports) are also called "config forms". The form report is considered a non-tabular report. To a developer, it resembles a report action.

Developing a form report is similar to developing a plain tabular report, and you should follow the same basic procedures. There are a few important differences. To create a form report, you must:

- Extend CloupiaNonTabularReport.
- Use getReportType, and return REPORT_TYPE_CONFIG_FORM.
- Extend CloupiaPageAction.
- isManagementReport returns True.

Procedure

Step 1
To extend CloupiaNonTabularReport, follow the example provided here:

Example:

```java
public class DummyFormReport extends CloupiaNonTabularReport {
    private static final string NAME = "foo.dummy.form.chart.report";
    private static final string LABEL = "Dummy Form Report";
}
```

Step 2
Set up getReportType and isManagementReport, referring to this code snippet:
Make sure that isManagementReport returns true. If you return false, the UI will not show your form.

Example:

```java
@Override
public int getReportType() {
    return ReportDefinition.REPORT_TYPE_CONFIG_FORM;
}

@Override
public boolean isManagementReport() {
    return true;
}
```

Step 3
Extend the CloupiaPageAction class to define an action that will trigger the form layout.
For the form report, the Report implementation class will be different from other report implementations.

Example:

```java
@Override
public void definePage(Page page, ReportContext context) {
    // This is where you define the layout of your action.
    // The easiest way to do this is to use this "bind" method.
    // Since I already have my form object, I just need to provide
    // a unique ID and the POJO itself. The framework will handle all the other details.
    page.bind(formId, DummyFormReportObject.class);
    // A common request is to hide the submit button which normally comes for free with
    // any form. In this particular case, because this form will show as a report,
}
I would like to hide the submit button, which is what this line demonstrates
page.setSubmitButton("");
}

When the user clicks the Submit button in the UI, the method validatePageData (shown in the following step) is called.

Step 4 Set up validatePageData as shown in this code example:

Example:

```java
@Override
public int validatePageData(Page page, report Context context,
WizardSession session) throws exception {
   return PageIf.STATUS_OK;
}
```

For additional examples of successful form report code, refer to:

Developing Schedule Tasks

This chapter contains the following sections:

- About Schedule Tasks, page 53

About Schedule Tasks

If you need to develop a purge task or aggregation task, or some other some kind of repeatable task, you can use the Schedule Task framework, which includes the following components:

- AbstractScheduleTask
- AbstractCloupiaModule

AbstractScheduleTask

Your task logic should be placed in the execute() method of this class. Provide your module ID and a string that describes this task to get started. You must to provide your own module ID, or the module will not be registered properly.

Adding/Removing Schedule Tasks

AbstractCloupiaModule has an add and remove schedule task API. Typically, in the onStart() implementation of your AbstractCloupiaModule, you would instantiate your tasks and register them with the add methods. However, these APIs are not always accessible. To retrieve the instance of the module that gives you access to these APIs, use the following code snippet (appropriately modified) anywhere in your code:

```java
YourModule module = (YourModule) FeatureContainer.getInstance().getModuleById(yourModuleID);
```
Developing Schedule Tasks

About Schedule Tasks
Developing a Trigger Condition

This chapter contains the following section:

- About Adding New Trigger Conditions, page 55
- Adding New Trigger Conditions, page 56

About Adding New Trigger Conditions

To create a trigger for a specific new purpose, you must have a trigger condition that is correctly defined. If an appropriate trigger condition does not already exist, you have to implement it. Likewise, if the appropriate and necessary components of the condition are not yet defined, then you can implement them using the information provided here. In the Create Trigger Wizard (found under Policies > Orchestration > Triggers, at the Specify Conditions step, you should have the options available to set up the new trigger condition.

A trigger is composed of two components:

- An implementation of `com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredContextIf`.
- At least one implementation of `com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoredParameterIf`.

The `MonitoredContextIf` is supposed to describe the object that is to be monitored and supply a list of references to the object. When you use the Edit Trigger > Specify Conditions element of the Wizard, you should see controls and related options that allow you to select the object and the references to it. For example, the `MonitoredContextIf` might be used to monitor the "Dummy Device" objects and to return a list of all the Dummy Devices available.

The `MonitoredParameterIf` is used in the definition of a trigger condition as follows:

- It provides the specific parameter to be examined. For example, it could be a parameter representing the status of the particular Dummy Device (for example, ddTwo) as defined by the `MonitorContextIf`.
- It supplies the operations that can be applied to the parameter. Typical operations include, for example:
  - less than
  - equal to
  - greater than
(The appropriate operations depend on the implementation.)

- It supplies a list of values, each of which can be logically compared against the parameter to activate the trigger.

So, for example, a trigger condition such as "Dummy Device ddTwo Status is down" can be logically tested as a condition. If the monitored Status parameter renders the statement True, the trigger condition is met.

## Adding New Trigger Conditions

### Before You Begin

Refer to the Open Automation javadocs for details on the implementation of the interface.

### Procedure

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Implement a <strong>MonitoredContextIf</strong> and all the applicable <strong>MonitoredParameterIfs</strong>. Refer to the sample code for an example.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2</td>
<td>Register the trigger condition into the system. <strong>com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoringTriggerUtil</strong> has a static method for this purpose.</td>
</tr>
<tr>
<td></td>
<td>a) Group your <strong>MonitoredContextIf</strong> and its <strong>MonitoredParameterIfs</strong> together into a <strong>com.cloupia.service.cIM.inframgr.thresholdmonitor.MonitoringTrigger</strong>.</td>
</tr>
<tr>
<td></td>
<td>b) Register the monitoring trigger with the utility.</td>
</tr>
<tr>
<td></td>
<td>For examples, see the following classes:</td>
</tr>
<tr>
<td></td>
<td>- <code>FooModule.java</code></td>
</tr>
<tr>
<td></td>
<td>- <code>MonitorDummyDeviceStatusParam.java</code></td>
</tr>
<tr>
<td></td>
<td>- <code>MonitorDummyDeviceType.java</code></td>
</tr>
</tbody>
</table>

### What to Do Next
CHAPTER 13

Developing New Resource Limits

This chapter contains the following sections:

- About New Resource Limits, page 57
- Associating a Resource with a Group, page 57
- Developing a Resource Computer, page 58
- Registering the New Resource, page 58
- Testing Your New Resource Limit Check, page 58

About New Resource Limits

You can introduce new resource types to be subjected to limit checks. For example, if you add device support for a new storage device, you might have limit checks for the number of logical unit numbers (LUNs) used by a group. You might want to be more granular and limit the number of gigabytes used per LUN by a group. Resource limits are associated per group, so whatever resource type that you add to the system, you must associate it with a group.

Introducing new resource limit types involves two procedures:

- Associating the new resource type to a group.
- Developing a computer that calculates the usage of the new resource for a group.

Associating a Resource with a Group

The new resource type that you want limited must be associated with a group. We recommend that you develop a new user interface (UI) action that allows the user to associate a group with the resource. To do so, you develop a report with a generic UI action. Similar to what you did in the Developing Reports Using POJO and Annotations, on page 42, but you must ensure that the form object includes a reference to the UserGroupsLOVProvider.NAMELOV provider. This identifies the group with which the user is associating the resource. You must also decide how to track the usage of the resource.
Developing a Resource Computer

After associating the resource with a group, you must develop a resource computer to compute the resource limit for the group. The resource computer implements the com.cloupia.service.cIM.inframgr.resourcecomputer.ResourceCountComputerIf interface to pass the group details required for the resource limit calculation. Using the association between resource and group, you can determine which group is using which resources and can calculate the full amount of resource being used by the group.

For more information on how to use this interface, refer to the sample code: com.cloupia.feature.foo.resourceComputer.DummyVLANResourceComputer.

Registering the New Resource

You need to register your resource computer in AbstractCloupiaModule. For more information, see FooModule.java in the sample code. You must provide a string to uniquely identify the resource type, a description of the resource type so it can be displayed in the UI, and the corresponding resource computer to be used.

Before You Begin

• Associate a resource to a group.
• Compute a resource limit.

Testing Your New Resource Limit Check

To verify that a newly introduced resource type is being checked properly, perform the following steps in the Cisco UCS Director UI.

• Verify that the resource type that you introduced is listed as a resource that can be limited.
• Set a resource limit for your new resource.
• Verify that your resource computer is working properly by checking the Resource Limits report.

Procedure

Step 1 Choose Administration > Users and Groups > User Groups > (Your Group), then click Edit Resource Limits. In the Resource Limit dialog box, select Enable Resource Limits. The Resource Limits dialog box expands into a page that shows all resources that can be limited. The resource type that you introduced should be visible under the Physical Resource Limits heading.
The sample code shows a Dummy VLAN resource. If you have implemented this resource, it should be present in the listing.

**Step 2**  
In the Resource Limits listing, set a limit for the resource type that you introduced, and click **Save**.

**Step 3**  
Check the **Resource Limits** report table. This is located under **Organizations > Summary > (Your Group) > Resource Limits**. After choosing the Group, click the **Resource Limits** tab.  
In the **Resource Limits** tab, look for a row of information about your resource.

The limit that you set is displayed. You can put the resource into usage (and take it to the usage limit) and recheck this report to verify that it is working as expected.
Testing Your New Resource Limit Check
Developing Report Actions

This chapter contains the following sections:

- About Report Actions, page 61
- Registering Actions with Report, page 62

About Report Actions

When developing reports, you can also include actions to go with those reports.

Developing a report action requires the following things:

- A form object—Describes what inputs the user can provide.
- A handler—Handles setting data in the form and what happens when the user hits the submit button.

This structure is similar to workflow tasks because workflows use the same framework. The difference is that your handler will extend from a different class.

Form Object

The form object is a POJO with JDO and FormField annotations. It should look very similar to the config objects that you developed for your workflow tasks. The only difference is that there is no need to implement any interface. You might want to use the JDO annotations if the form has data that you want persisted; they make it easier to retrieve and persist the POJO from the database. Use the FormField annotations to specify which user interface (UI) control to display to the user.

Form Handler

You should place the logic for your report action in a handler. The handler extends com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaPageAction.

The form handler includes several methods that you might want to override:

- definePage—Defines the layout of the form. You want to bind your form object to the page, and the UI will generate a form as defined by your form object.
- loadDataToPage—Handles loading data that you might want to show in the form when it is first displayed to the user.
• validatePageData—Handles whatever logic that you want to execute when the user presses the submit button.

For examples of the form object and the form handler, see the com.cloupia.feature.foo.actions.SimpleDummyAction in the sample code.

Registering Actions with Report

After completing the development of actions, you must associate them with a report, such as a report of type com.cloupia.service.cIM.inframgr.reports.simplified.CloupiaReportWithActions. To associate them with a report, you must implement a getActions method and provide instances of your CloupiaReportAction.
Developing CloudSense Reports

This chapter contains the following sections:

- Developing CloudSense Reports, page 63

Developing CloudSense Reports

Each CloudSense report consists of two components:

- An XML descriptor file
- The actual implementation of the report in JavaScript

The XML descriptor file requires the following:

- A unique string to identify the report
- A description of the report that used to display in the user interface (UI)
- The name of the implementation file for the report
- The context that should be used when generating the report

The implementation of CloudSense reports in JavaScript is described in detail in the Cisco UCS Director documentation.

You should place all CloudSense report-related files in a folder called cloudsense. The Zip file that you upload through Open Automation must contain the cloudsense folder. See the sample module for more information about the folder structure.
Developing CloudSense Reports
Change Tracking API

This chapter contains the following sections:

- Change Tracking API, page 65

Change Tracking API

You can use the Change Tracking API to track changes that are made through their module and to record the changes in the database.

The constructor is ChangeTrackingAPI.

```java
package com.cloupia.feature.foo.scheduledTasks;
import org.apache.log4j.Logger;
import com.cloupia.feature.foo.FooModule;
import com.cloupia.model.cIM.ChangeRecord;
import com.cloupia.service.cIM.inframgr.AbstractScheduleTask;
import com.cloupia.service.cIM.inframgr.FeatureContainer;
import com.cloupia.service.cIM.inframgr.cmdb.ChangeTrackingAPI;

/**
 * This is a simple example demonstrating how to implement a scheduled task. This task is executed
 * every 5 mins and simply makes a logging statement and increments the number of times it's been
 * executed. It removes itself from the system once it has been executed twice. It also demonstrates how
 * you can use the change tracking APIs to track changes made to the system.
 *
 */
public class DummyScheduleTask extends AbstractScheduleTask {
    private static Logger logger = Logger.getLogger(DummyScheduleTask.class);
    private int numTimesExecuted = 0;
    private static final long TWO_MINS = 60*1000*2;
    private static final int MAX_TIMES_EXECUTED = 2;

    public DummyScheduleTask() {
        super("foo");
    }

    @Override
    public void execute(long lastExecution) throws Exception {
        // Your implementation here
    }
}
```
logger.info("vxvxvxvxvx - dummyTask has been executed " + numTimesExecuted + " times.");
numTimesExecuted++;

if (numTimesExecuted == MAX_TIMES_EXECUTED) {
    logger.info("vxvxvxvxvx - removing dummyTask");
    FooModule module = (FooModule) FeatureContainer.getInstance().getModuleById("foo");
    //NOTE: Use getTaskName() and NOT getScheduleTaskName(), it's really important
    //We distinguish the two: getTaskName is used internally by the system, where we do
    //some extra stuff to ensure uniqueness of the task name (prepend moduleID), so we need
to
    //make sure to use this when removing tasks!
    module.removeScheduleTask(this.getTaskName());
    //use the static ChangeTrackingAPI to create an instance of ChangeRecord, these are just
    //values you'd like have
    ChangeRecord rec = ChangeTrackingAPI.create("openAutoDeveloper",
        ChangeRecord.CHANGE_TYPE_DELETE, "Dummy Task removed from System",
        "foo dummy task");
    //insert the record like so
    ChangeTrackingAPI.insertRecord(rec);
}

@Override
public long getFrequency() {
    return TWO_MINS;
}

@Override
protected String getScheduleTaskName() {
    //usually good idea to name your task something descriptive
    return "dummyTask";
}
Appendix A

This chapter contains the following sections:

- Existing List of Value Tables, page 67

Existing List of Value Tables

The following table lists existing tabular reports of available lists of values.

<table>
<thead>
<tr>
<th>Report: List/Provider Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vdiMcsCatalogAllocationTypeList</td>
<td>VDI MCS Catalog Allocation Type Selector</td>
</tr>
<tr>
<td>netAppClusterPortAssIfGroupProvider</td>
<td>NetApp Cluster Port Associated IfGroup Selector</td>
</tr>
<tr>
<td>catalogList</td>
<td>Catalog Selector</td>
</tr>
<tr>
<td>DiskSizesList</td>
<td>Disk Size Selector</td>
</tr>
<tr>
<td>vdcProfileList</td>
<td>vDC Profile Selector</td>
</tr>
<tr>
<td>vmwareCloudNamesList</td>
<td>VMware Account Selector</td>
</tr>
<tr>
<td>protocolList</td>
<td>NetApp vFiler Protocol List</td>
</tr>
<tr>
<td>MemorySizesList</td>
<td>Memory Size Selector</td>
</tr>
<tr>
<td>vdiVdcList</td>
<td>VDI VDC Selector</td>
</tr>
<tr>
<td>emcRAIDTypesListProvider</td>
<td>EMC RAID Type Selector</td>
</tr>
<tr>
<td>portGroupTypeList</td>
<td>VMware Portgroup Type</td>
</tr>
<tr>
<td>ucsNetworkPolicyList</td>
<td>Cisco UCS Network Policy</td>
</tr>
<tr>
<td><strong>Report: ListProvider Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>--------------------------------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>iGroupTypeList</td>
<td>NetApp Initiator Group Type</td>
</tr>
<tr>
<td>vdiCatalogList</td>
<td>VDI Catalog Selector</td>
</tr>
<tr>
<td>hpServerBootActionTypes</td>
<td>HP Boot Mode</td>
</tr>
<tr>
<td>netappAllClusterAssocNonAssocIfGroupList</td>
<td>NetApp Cluster All IfGroup Selector</td>
</tr>
<tr>
<td>dfmStorageServiceList</td>
<td>NetApp OnCommand Storage Services</td>
</tr>
<tr>
<td>NetworkDevicePortProfileModeList</td>
<td>Networking Port Profile Mode</td>
</tr>
<tr>
<td>ucsServiceProfileList</td>
<td>Cisco UCS Service Profile Selector</td>
</tr>
<tr>
<td>hostNodeList</td>
<td>Host Node Selector</td>
</tr>
<tr>
<td>vdiMcsCatalogTypeList</td>
<td>VDI MCS Catalog Type Selector</td>
</tr>
<tr>
<td>netappAllClusterVLANList</td>
<td>NetApp Cluster vLAN Selector</td>
</tr>
<tr>
<td>vdcList</td>
<td>vDC Selector</td>
</tr>
<tr>
<td>netappAllClusterAggrList</td>
<td>NetApp Cluster Aggregate Selector</td>
</tr>
<tr>
<td>NetworkDeviceBasicList</td>
<td>Networking Device</td>
</tr>
<tr>
<td>netappAllClusterPortList</td>
<td>NetApp Cluster Port Selector</td>
</tr>
<tr>
<td>MSPGroupList</td>
<td>MSP Group</td>
</tr>
<tr>
<td>sizeUnitList</td>
<td>NetApp Size Units</td>
</tr>
<tr>
<td>amazonVMList</td>
<td>Amazon VM Selector</td>
</tr>
<tr>
<td>ec2AccountList</td>
<td>Amazon Cloud Selector</td>
</tr>
<tr>
<td>ucsBladeList</td>
<td>Cisco UCS Server</td>
</tr>
<tr>
<td>emcSizeUnits</td>
<td>EMC Size Units</td>
</tr>
<tr>
<td>vdiMemorySizesList</td>
<td>VDI Memory Size Selector</td>
</tr>
<tr>
<td>VMwareDVSUplinkPortgroupList</td>
<td>VMware DVSUplink Portgroup</td>
</tr>
<tr>
<td>hpServerAutoPowerActionTypes</td>
<td>Hewlett Packard (HP) Auto Power Mode</td>
</tr>
<tr>
<td>hpServerBootSourceAction</td>
<td>HP Boot Source</td>
</tr>
<tr>
<td>Report: ListProvider Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>hpServerPowerSaverActionTypes</td>
<td>HP Power Saver Mode</td>
</tr>
<tr>
<td>hpServerPowerSaverActionTypes</td>
<td>HP Power Saver Mode</td>
</tr>
<tr>
<td>osTypeList</td>
<td>NetApp OS Type</td>
</tr>
<tr>
<td>vdiCPUsList</td>
<td>VDI CPU Selector</td>
</tr>
<tr>
<td>vscResizeDatastoreList</td>
<td>NetApp VSC Resize Datastore Selector</td>
</tr>
<tr>
<td>UcsAccountList</td>
<td>Cisco UCS Account</td>
</tr>
<tr>
<td>containerProvider</td>
<td>Service Container</td>
</tr>
<tr>
<td>amazonVMActions</td>
<td>Generic VM Action Selector</td>
</tr>
<tr>
<td>dfmGroupList</td>
<td>NetApp OnCommand Groups</td>
</tr>
<tr>
<td>hpServerPowerActionTypes</td>
<td>HP Server Power Mode</td>
</tr>
<tr>
<td>kvmVMActions</td>
<td>KVM VM Action Selector</td>
</tr>
<tr>
<td>netappAllClusterNodeList</td>
<td>NetApp Cluster Node Selector</td>
</tr>
<tr>
<td>vdiAccountList</td>
<td>VDI Account Selector</td>
</tr>
<tr>
<td>NetworkDevicePortProfileTypeList</td>
<td>Networking Port Profile Type</td>
</tr>
<tr>
<td>ec2VolumeSizeList</td>
<td>EC2 Volume Size Selector</td>
</tr>
<tr>
<td>userGroupsLOV</td>
<td>User Group</td>
</tr>
<tr>
<td>vmwarePortGroupLoVProvider</td>
<td>VMware Port Group</td>
</tr>
<tr>
<td>kvmVMList</td>
<td>KVM VM Selector</td>
</tr>
</tbody>
</table>
Appendix A

Existing List of Value Tables
Appendix B

• Report Context Types and Report Context Names, page 71

Report Context Types and Report Context Names

About Report Context Data

Use this list to find and to specify the context data of a report. It provides the Report Context Type, a unique ID number associated with each report context, followed by the Report Context Name. For basic information about reports and contexts, see the Cisco UCS Director REST API Guide.

0 global
1 cloud
2 hostnode
3 vm
4 action
5 cluster
6 services
7 group
8 user
9 event
10 global_admin
11 catalog
12 service_request
13 user_workflow
14 amz_dep_policy
15 rackspace_policy
16 compute_chassis
17 compute_server
Appendix B

Report Context Types and Report Context Names

18 compute_io_card
19 compute_fbi
20 compute_fbi_port
21 vdc
22 vdc_reports
23 datacenter
24 alldatacenters
25 service_profile
26 server_pool
27 vm_snapshot
28 managed_report
29 cloudsense_partner
30 cloudsense_customer
31 cloudsense_partner_admin
32 cloudsense_admin
33 storage_volumes
34 custom_actions
35 storage_aggregates
36 storage_disks
37 ucs
38 pari
39 pari_device
40 storage_accounts
41 host_account
42 network
43 network_device
44 luns
45 network_switch_port
46 net_device_generic
47 net_device_n1k
48 net_device_fab_ic
49 net_device_n5k
50 ucs_storage_policy
51 ucs_network_policy
52 msp
53 ucs_org
54 ucs_policies
55 ucs_mac
56 ucs_boot_policy
57 netapp_dfm_prov_policy
58 netapp_dfm_prot_policy
59 netapp_dfm_service_policy
60 netapp_dfm_vfiler_template_policy
61 ucs_sp_template
62 netapp_dfm_dataset
63 rhev_datacenter
64 rhev_storage_domain
65 rhev_user
66 rhev_template
67 netapp_filer
68 ucs_wwnn
69 ucs_wwpn
70 ucs_uuid
71 ucs_iomodule
72 ucs_iomodule_port
73 netapp_v_filer
74 netapp_vfiler_volume
75 netapp_vfiler_lun
76 ucs_service_profile
77 ucs_service_profile_template
78 vmware_port_group
79 ucs_portchannel
80 netapp_initiator_group
81 netapp_initiator
82 netapp_vfiler_initiator_group
83 netapp_vfiler_initiator
84 kvm_dep_policy
85 vvd
86 ocap_module
87 hp
88 hp_server
89 ocap_module_report
Report Context Types and Report Context Names

90 ocap_module_file
91 ocap_module_changelog
92 ocap_module_workflowtasks
93 ocap_module_workflowinout
94 ocap_module_schedjobs
95 ocap_module_cloudsense
96 ucs_mac_block
97 compute_server_adapter_unit
98 ucs_locale
99 netapp_dfm_group
100 ucs_sp_vhba
101 ucs_sp_vnic
102 ucs_vniectemplate
103 net_device_qos_policy
104 netapp_ipspace
105 netapp_interface
106 ucs_vlan
107 vmware_network_vswitch
108 vmware_network_dvswitch
109 vmware_network_dvportgroup
110 vmware_network_vmknic
111 datastore
112 net_device_vlan
113 net_device_vsan
114 net_device_interface
115 net_device_port_profile
116 net_device_zone
117 storage_ip_proto_policy
118 standalone_rack_server_account
119 cimc_boot_definition
120 net_device_asa
121 net_device_asa_context
122 netapp_cifs_share
123 netapp_vfiler_cifs_share
124 netapp_qtree
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>126</td>
<td>netapp_vfiler_qtree</td>
</tr>
<tr>
<td>127</td>
<td>xenvdi_desktop</td>
</tr>
<tr>
<td>128</td>
<td>xenvdi_catalog</td>
</tr>
<tr>
<td>129</td>
<td>xenvdi_desktop_group</td>
</tr>
<tr>
<td>130</td>
<td>vvd_studio_wizard</td>
</tr>
<tr>
<td>131</td>
<td>vvd_studio_wizard-page</td>
</tr>
<tr>
<td>132</td>
<td>cimc_server_cpu</td>
</tr>
<tr>
<td>133</td>
<td>cimc_server_memory</td>
</tr>
<tr>
<td>134</td>
<td>cimc_server_pci_adapters</td>
</tr>
<tr>
<td>135</td>
<td>cimc_server_psu</td>
</tr>
<tr>
<td>136</td>
<td>cimc_server_fan</td>
</tr>
<tr>
<td>137</td>
<td>cimc_server_network_adapter</td>
</tr>
<tr>
<td>138</td>
<td>cimc_server_vhba</td>
</tr>
<tr>
<td>139</td>
<td>cimc_server_vhba_bootable</td>
</tr>
<tr>
<td>140</td>
<td>storage_cluster_controller</td>
</tr>
<tr>
<td>141</td>
<td>storage_cluster_account</td>
</tr>
<tr>
<td>142</td>
<td>storage_cluster_node</td>
</tr>
<tr>
<td>143</td>
<td>storage_cluster_vserver</td>
</tr>
<tr>
<td>144</td>
<td>storage_cluster_volume</td>
</tr>
<tr>
<td>145</td>
<td>cimc_server_vnic</td>
</tr>
<tr>
<td>146</td>
<td>tier3_group</td>
</tr>
<tr>
<td>147</td>
<td>emc_vnx_account</td>
</tr>
<tr>
<td>148</td>
<td>xenvdi_identity_pool</td>
</tr>
<tr>
<td>149</td>
<td>storage_cluster_lun</td>
</tr>
<tr>
<td>150</td>
<td>storage_cluster_igroup</td>
</tr>
<tr>
<td>151</td>
<td>storage_cluster_logInf</td>
</tr>
<tr>
<td>152</td>
<td>storage_cluster_initiator</td>
</tr>
<tr>
<td>153</td>
<td>storage_cluster_Qtree</td>
</tr>
<tr>
<td>154</td>
<td>emc_vnx_file_system</td>
</tr>
<tr>
<td>155</td>
<td>emc_vnx_data_mover</td>
</tr>
<tr>
<td>156</td>
<td>emc_vnx_mount</td>
</tr>
<tr>
<td>157</td>
<td>emc_vnx_cifs_share</td>
</tr>
<tr>
<td>158</td>
<td>emc_vnx_volume</td>
</tr>
<tr>
<td>159</td>
<td>emc_vnx_nfs_export</td>
</tr>
<tr>
<td>160</td>
<td>emc_vnx_storage_pool</td>
</tr>
<tr>
<td>161</td>
<td>scm_workspace</td>
</tr>
</tbody>
</table>
Appendix B

Report Context Types and Report Context Names

162 scm_config_account
163 emc_vnx_cifs_server
164 emc_vnx_dns_domain
165 emc_vnx_network_interface
166 emc_vnx_logical_net_device
167 net_device_brocade_300
168 vmware_network_nic
169 infra_load_balancer
170 emc_vnx_lun
171 emc_vnx_storage_pool_for_block
172 emc_vnx_raid_group
173 emc_vnx_lun_folder
174 emc_vnx_storage_group
175 emc_vnx_host
176 emc_vnx_host_initiator
177 emc_vnx_storage_processor
178 ipmi_account
179 ipmi_server
180 compute_accounts
181 hyperv_library_server
182 net_device_brocade_nos_vdx
183 vmware_image_context
184 netapp_snapmirror
185 netapp_vfiler_snapmirror
186 netapp_snapmirror_destination
187 netapp_vfiler_snapmirror_destination
188 netapp_snapmirror_schedule
189 netapp_vfiler_snapmirror_schedule
190 infra_compute_servers
191 net_device_brocade_nos
192 net_device_brocade_fos
193 datacenter_ucs_account
194 ucs_iqn_pool
195 resource_pool
196 dc_ucs_account_server
197 per_rack_server_account
198 dc_account_hp_server
199 per_account_ipmi_server
200 per_dc_ipmi_account
201 per_dc_hp_account
202 per_dc_cimc_account
203 external_service_request
204 net_device_cisco_nxos
205 net_device_cisco_nxos_n7k
206 storage_cluster_port
207 storage_cluster_ifgroup
208 storage_cluster_vlan
209 external_sr_chargeback
210 resources_chargeback
211 ucs_service_profile_bootpolicy
212 ucs_rack_mount_server
213 ucs_fabric_extender
214 ucs_fan_module
215 work_order
216 resource_alert
217 hyperv_image_context
218 hyperv_snapshot_context
219 storage_cluster_exportrule
220 ucs_fan
221 ucs_io_module_port
222 ucs_psu
223 ucs_server_dce
224 ucs_server_hba
225 ucs_server_nic
226 ucs_server_cpu
227 ucs_server_memory_unit
228 ucs_server_disk
229 ucs_vsan
230 ucs_lan_port_channel
231 ucs_fi_fc_port
232 custom_feature
233 ucs_vhbatemplate
Appendix B

Report Context Types and Report Context Names

234 ucs_vlan_group
235 emc_vmax_device
236 emc_vmax_account
237 emc_vmax_igroup
238 emc_vmax_portGroup
239 emc_vmax_storageGroup
240 emc_vmax_masking_view
241 ucs_lan_conn_policy
242 ucs_san_conn_policy
243 ucs_storage_conn_policy
244 ucs_storage_iGroup
245 ucs_decommissioned_server
246 hyperv_cluster_csv
247 hyperv_logical_network
248 hyperv_logicalnework_def
249 hyperv_vmnetwork_subnet
250 hyperv_vm_network
251 hyperv_logical_switch
252 hyperv_file_share
253 hyperv_cluster_available_nodes
254 ucs_sp_rename
255 hyperv_logicalnework_def_subnet
256 hyperv_storage_file_server
257 hyperv_storage_array
258 hyperv_storage_provider
259 hyperv_storage_classifications
260 hyperv_storage_pool
261 hyperv_storage_pool_per_array
262 vnmc_datacenter_account
263 vnmc_account
264 ucs_fault_suppress_task
265 hyperv_host_group
266 vnmc_tenant
267 vnmc_zone
268 vnmc_acl_policy_rule
269 vnmc_vdc
270 vnmc_vapp
271 vnmc_tier
272 net_device_n3k
273 net_device_mds
274 net_device_cisco_nxos_n7k_vdc
275 emc_vmax_director
276 hyperv_native_uplink_pp
277 hyperv_native_vna_pp
278 hyperv_static_ip_pool
279 hyperv_port_classification
280 hyperv_vm_network_adapter
281 emc_vmax_thin_pool
282 emc_vmax_storage_tier
283 storage_cluster_aggregates
284 net_device_ios
285 emc_vmax_fast_policy
286 emc_vmax_storage_tier
287 netapp_cluster_cron_job
288 netapp_cluster_snapshot_policy
289 netapp_cluster_snapshot_policy_schedule
290 ucs_local_disk_config_policy
291 emc_vnx_meta_lun
292 emc_vnx_block_account
293 emc_vnx_file_account
294 netapp_cluster_volume_snapshot
295 netapp_cluster_vserver_volume_cifs
296 ucs_discovered_server
297 ucs_central
298 ucs_central_domain_group
299 ucs_central_compute_system
300 net_device_cisco_nxos_n7k_vdc_storage
301 cluster_vserver_domain
302 net_device_n9k
303 cimc_server_storage_adapter
304 cimc_server_storage_adapter_summary
305 context_type_cimc_server_storage_adapter_physical
Report Context Types and Report Context Names

306 context_type_cimc_server_storage_adapter_virtual
307 cimc_server_processor_unit
308 cimc_server_pci_adapter_unit
309 cimc_server_network_adapters
310 cimc_server_network_adapters_eth
311 cimc_server_new_psu
312 cluster_vserver_Ip_Host_Mapping
313 rack_server_api_supported
314 vnmc_vsg_policy
315 netapp_cluster_portset
316 cluster_vserver_sis_policy
317 cimc_server_vic_adapter
318 cimc_server_vic_adapter_vhba
319 cimc_server_vic_adapter_vnic
320 cluster_wwpn_alias
321 system_task
322 system_task_history
323 remote_agent
324 agent_tasks
400 xyz_context
425 all_pods_physical_compute
426 ucs_central_org
427 pnsc_cs_profile
428 netapp_cluster_nfs_service
429 all_pods_physical_storage
430 all_pods_physical_network
431 ucs_central_accounts
432 multi_domain_managers
433 hyperv_host_adapter_ln
434 hyperv_vnetwork_hostadapter
435 netapp_cluster_vserver_peer
436 netapp_cluster_snapmirror
437 pnsc_policy_set
438 netapp_cluster_snapmirror_policy
439 pnsc_policy_list
440 ucs_central_chassis
Appendix B

Report Context Types and Report Context Names

441 ucs_central_server
442 ucs_central_server_storage_controller
443 ucs_central_vnic_template
444 ucs_central_vhba_template
445 ucs_central_service_profile
446 ucs_central_service_profile_templ
447 ucs_central_wwpn
448 ucs_central_wwnn
449 ucs_central_mac
450 ucs_central_uuid
451 ucs_central_ippool
452 ucs_central_server_pool
453 ucs_central_lan_conn_pol
454 ucs_central_san_conn_pol
455 netapp_cluster_job
456 net_device_n6k
457 ucs_central_boot_policy
458 pnsc_compute_firewall
459 net_device_n1110
460 ucs_central_fan_module
461 netapp_cluster_vserver_routing_group
462 netapp_cluster_peer
463 ucs_central_vsan
464 ucs_central_vlan
465 ucs_central_rack_mount_server
466 ucs_iscsi_adapter_policy
467 ucs_network_control_policy
468 ucs_qos_policy
469 ucs_central_fex
470 whiptail_account
471 whiptail_initiator_group
472 whiptail_volume_group
473 whiptail_lun
474 whiptail_interface
475 whiptail_acela
476 whiptail_invicta
477 ucs_central_fc_adapter_policy
478 ucs_central_firmware_policy
479 ucs_central_maintenance_policy
480 ucs_central_server_pool_policy
481 ucs_central_server_pool_policy_qual
482 ucs_central_vnic_vhba_placement_policy
483 ucs_central_vhba_policy
484 ucs_central_vnic_policy
485 ucs_central_storage_policy
486 ucs_central_network_policy
487 ucs_central_local_disk_policy
488 ucs_central_iqn_pool
489 netapp_cluster_export_policy
490 ucs_central_local_service_profile
491 ucs_central_local_service_profile_templ
492 ucs_central_vnic
493 ucs_central_vhba
494 ucs_central_fabric_interconnect
495 network_static_ip_pool_policy
496 hyperv_storage_fileshare
497 hyperv_storage_lun
498 netapp_cluster_disk
499 hyperv_host_group_storage_pool
500 hyperv_host_group_storage_lun
501 pnsc_accounts
502 emc_vmax_datadev
503 emc_vmax_thindev
504 netapp_vlan_interface
90001 collector.data.collection.policy.report
90002 collector.data.collection.policy.associate.report
90003 DummyAccount.generic.infra.report.6000:2
90004 foo.dummy.drilldown.interface.report
90005 VMAX System Devices
90006 System Summary
90007 VMAX Tiers
90008 VMAX Symmetrix Devices
90009 VMAX Thin Devices
90010 VMAX Meta Devices
90011 VMAX Initiator Groups
90012 VMAX Initiators
90013 VMAX Storage Groups
90014 VMAX Port Groups
90015 VMAX Masking Views
90016 VMAX Thin Pools
90017 VMAX Fast Policies
90018 VMAX Fast Controller
90019 VMAX FAST Status
90020 WHIPTAIL System Summary
90021 WHIPTAIL SSR Report
90022 WHIPTAIL Bonds Report
90023 WHIPTAIL Virtual Interfaces Report
90024 WHIPTAIL VLANs Report
90025 WHIPTAIL Physical Interfaces Report
90026 WHIPTAIL iSCSI Settings Report
90027 WHIPTAIL SSNs Report
90028 WHIPTAIL RAID Health Report
90029 WHIPTAIL Volume Groups Report
90030 WHIPTAIL LUNs Report
90031 WHIPTAIL Initiator Groups Report
90032 WHIPTAIL FC Report
90033 cluster.system.tasks.policy.report
90034 foo.dummy.context.one
90033 cluster.system.tasks.policy.report
90034 foo.dummy.context.one