Overview

This chapter contains the following sections:

Cisco UCS Director, Custom Tasks, and CloupiaScript

Cisco UCS Director provides automated, profile-based provisioning, management, and reporting of infrastructure resources. Cisco UCS Director incorporates a powerful orchestration engine that enables complex operations on any element of your converged infrastructure, both physical and virtual. These operations are embodied in workflows, which are scripted sequences of individual tasks. Cisco UCS Director comes complete with a large library of tasks.

The tasks in Cisco UCS Director are written in CloupiaScript, a version of JavaScript with libraries that enable orchestration operations. With CloupiaScript, it is possible to embed scripts in workflow tasks and to write custom tasks.

You should be familiar with Cisco UCS Director and Cisco UCS Director Orchestrator to take advantage of the scripted examples in this Cookbook.

For information about installation and administration of Cisco UCS Director, see the Cisco UCS Director Administration Guide. For help using the Cisco UCS Director Orchestrator, see the Cisco UCS Director Orchestration Guide. For a reference to CloupiaScript classes and methods, see the CloupiaScript Javadoc included with the Cisco UCS Director Script Bundle.

Structure of an Example

Under a descriptive title, each example comprises the following sections:

Objective

What the example is designed to accomplish.

Context

When you would use the example, when you would not use it, and why.

Prerequisites

What conditions have to exist for the example to work.

Components

Which objects and methods are used in the example, and what the input variables represent.

Code

The example code.

Results

What output is expected from the example code.

Implementation

Notes on implementing the example, including what modifications might be necessary to implement it.

See Also

Related Examples

How to Use the Examples

This document is a collection of examples, recipes, if you will, for using CloupiaScript, a server-side scripting solution for use with Cisco UCS Director; Orchestrator. Like a cookbook, you can use this document in at least three ways:

- You can follow the examples as written (substituting your own variables, of course) to complete tasks without necessarily knowing everything about the steps you are following.
- You can use the examples as templates and adapt them to similar tasks in your work.
- You can study the examples to figure out “how things are done” in CloupiaScript and, along with the CloupiaScript Javadoc reference, generalize to using different methods for other tasks you need to script.

The examples are chosen to illustrate common use cases and are intended to facilitate all three of these modes of use.

Getting Inputs and Outputs of a Service Request

To view the input or output of a service request in Cisco UCS Director, do the following:

Step 1 Choose **Organizations** > **Service Requests**.

Step 2 On the **Service Requests** page, click **Service Requests**.

Step 3 Click the row with the service request that you want to view.

Step 4 From the **More Actions** drop-down list, choose **View Details**.
The **Service Request** screen displays the wokflow status, log, objects created and modified, and input/output of the service request.

Step 5 Scroll down the screen to view the input and output of the workflow and sub-workflow.

Step 6 Click **Close** to close the **Service Request** screen.

How to Access XML REST APIs

The following APIs are used to invoke XML REST APIs and perform basic operation in Cisco UCS Director:

- userAPIMoCreate—Invokes an HTTP POST XML REST APIs to perform the create operation.
- userAPIMoQuery—Invokes an HTTP GET XML REST API to perform the retrieve operation.
- userAPIMoUpdate—Invokes an HTTP PUT XML REST API to perform the update operation.
- userAPIMoDelete—Invokes an HTTP DELETE XML REST API to perform the delete operation.

userAPIMoCreate

The **userAPIMoCreate** API invokes any one of the HTTP POST XML REST APIs from Cisco UCS Director. The **userAPIMoCreate** API takes both a resource path and XML payload as input, and performs the HTTP Post operation on the given resource path.

Note The payload must be in XML format.

The following example shows how to invoke the user@CREATE XML REST API to create a service end-user:

```
importClass(javax.xml.parsers.DocumentBuilder);
importClass(javax.xml.parsers.DocumentBuilderFactory);
importClass(javax.xml.parsers.ParserConfigurationException);
importPackage(java.io);
importPackage(javax.xml.transform);
importPackage(javax.xml.transform.dom);
importPackage(javax.xml.transform.stream);
importClass(org.w3c.dom.CDATASection);
importClass(org.w3c.dom.Document);
importClass(org.w3c.dom.Element);
```

```
importClass(org.w3c.dom.Node);

logger.addInfo("Demo for userAPIMoCreate REST API");
logger.addInfo("Execute user@CREATE HTTP Post XML REST API through userAPIMoCreate");
//Create a service end-user with mandatory parameters
var resourcePath = "/user";
//Construct payload for user@CREATE
//Mandatory parameters to create a service end-user
var userRoleType = "Regular";
var userGroupId = "1";
var userName = "sdk_EU";
var userPassword = "paas@12345";
var userEmailId = "abc@xyz.com"
function executeUserAPIMoCreate() {
    var xmlPayload = constructPayload();
    logger.addInfo("XML payload string to create a Service End-User: " + xmlPayload);
    var response = ctxt.getAPI().userAPIMoCreate(resourcePath, xmlPayload);
    logger.addInfo("user creation response : "+response);
}
function constructPayload() {
    var dbf = DocumentBuilderFactory.newInstance();
    var builder;
    var xmlStr = null;
    builder = dbf.newDocumentBuilder();
    var doc = builder.newDocument();
    var cuicOperationRequest = doc.createElement("cuicOperationRequest");
    var payload = doc.createElement("payload");
    var addUserConfig = doc.createElement("AddUserConfig");
    var userType = doc.createElement("userType");
    userType.appendChild(doc.createTextNode(userRoleType));
    var userGroup = doc.createElement("userGroup");
    userGroup.appendChild(doc.createTextNode(userGroupId));
    var loginName = doc.createElement("loginName");
    loginName.appendChild(doc.createTextNode(userName));
    var password = doc.createElement("password");
    password.appendChild(doc.createTextNode(userPassword));
    var confirmPassword = doc.createElement("confirmPassword");
    confirmPassword.appendChild(doc.createTextNode(userPassword));
    var userContactEmail = doc.createElement("userContactEmail");
    userContactEmail.appendChild(doc.createTextNode(userEmailId));
    addUserConfig.appendChild(userType);
    addUserConfig.appendChild(userGroup);
    addUserConfig.appendChild(loginName);
    addUserConfig.appendChild(password);
    addUserConfig.appendChild(confirmPassword);
    addUserConfig.appendChild(userContactEmail);
    doc.appendChild(cuicOperationRequest);
    xmlStr = getXMLString(addUserConfig);
    var cdata = doc.createCDATASection(xmlStr);
    payload.appendChild(cdata);
    cuicOperationRequest.appendChild(payload);
    xmlStr = null;
    xmlStr = getXMLString(doc);
    return xmlStr;
}
function getXMLString(node) {
    var xmlStr = null;
    try {
        var domSource = new DOMSource(node);
        var writer = new StringWriter();
        var result = new StreamResult(writer);
        var transformerFactory = TransformerFactory.newInstance();
        var transformer = transformerFactory.newTransformer();
        transformer.transform(domSource, result);
        xmlStr = writer.toString();
    } catch (ex) {
        throw ex;
    }
    return xmlStr;
}
try {
    executeUserAPIMoCreate();
} catch (e) {
```

```
    logger.addError("Error:" + e);
    ctxt.setFailed("Task failed to execute 'userAPIMoCreate' REST API");
    ctxt.exit();
}
```

userAPIMoQuery

The **userAPIMoQuery** API invokes any one of the HTTP GET XML REST APIs from Cisco UCS Director. The **userAPIMoQuery** API takes a resource path as input and returns the resources in XML format. Then, the XML response is parsed to retrieve the required information.

The following example shows how to invoke the user@READ XML REST API to retrieve a list of users from Cisco UCS Director:

```
importClass(java.io.StringReader);
importClass(javax.xml.parsers.DocumentBuilder);
importClass(javax.xml.parsers.DocumentBuilderFactory);
importClass(org.w3c.dom.Document);
importClass(org.w3c.dom.NodeList);
importClass(org.w3c.dom.Node);
importClass(org.xml.sax.InputSource);

logger.addInfo("Demo for userAPIMoQuery REST API");
logger.addInfo("Execute user@READ HTTP GET XML REST API through userAPIMoQuery");

//Read all users
//To read a specific user, the resource path is '/user/loginName'
var resourcePath = "/user";
function executeUserAPIMoQuery() {
    //Second and third parameters are not applicable
    var response = ctxt.getAPI().userAPIMoQuery(resourcePath, null, null);
    logger.addInfo("you got the response");
    logger.addInfo("Read all users response : " + response);
    getUserDetails(response);
}
function getUserDetails(response) {
    var dbf = DocumentBuilderFactory.newInstance();
    var builder;
    var nodeList = null;
    try {
        builder = dbf.newDocumentBuilder();
        var doc = builder.parse(new InputSource(new StringReader(response)));
        nodeList = doc.getElementsByTagName("cuicOperationStatus");
        var operationStatus = null;
        if (nodeList != null && nodeList.getLength() > 0) {
            operationStatus = nodeList.item(0).getTextContent();
            logger.addInfo("cuicOperationStatus = " + operationStatus);
        }
        if (operationStatus.equals("0")) {
            logger.addInfo("Successfully Read all UCSD users!!");
            nodeList = null;
            nodeList = doc.getElementsByTagName("user");
            if (nodeList != null && nodeList.getLength() > 0) {
                for (var i = 0; i < nodeList.getLength(); i++) {
                   logger.addInfo("*******Reading details for User" + (i + 1) + "*******");

                    printUserDetails(nodeList.item(i));
                    logger.addInfo("*******End of User details for user" + (i + 1) +
"*******");
                }
            }
        } else {
            var errorMsg = null;
            nodeList = null;
            nodeList = doc.getElementsByTagName("errorMessage");
            if (nodeList != null && nodeList.getLength() > 0) {
                errorMsg = nodeList.item(0).getTextContent();
            }
            logger.addInfo("cuicOperationStatus = " + operationStatus);
```

```
            logger.addInfo("errorMessage = " + errorMsg);
        }
    } catch (ex) {
        throw ex;
    }
}
function printUserDetails(node) {
    var nodeList = null;
    nodeList = node.getChildNodes();
    if (nodeList != null && nodeList.getLength() > 0)
        for (var i = 0; i < nodeList.getLength(); i++) {
            var currentNode = nodeList.item(i);
            if (currentNode.getNodeType() == Node.ELEMENT_NODE) {
                logger.addInfo(currentNode.getNodeName());
                printUserDetails(currentNode);
            }
            if (currentNode.getNodeType() == Node.TEXT_NODE)
                logger.addInfo(":"+currentNode.getTextContent());
        }
}
try {
    executeUserAPIMoQuery();
} catch (e) {
    logger.addError("Error:" + e);
    ctxt.setFailed("Task failed to execute 'userAPIMoQuery' REST API");
    ctxt.exit();
}
```

userAPIMoUpdate

The **userAPIMoUpdate** API invokes any one of the HTTP PUT XML REST APIs from Cisco UCS Director. The **userAPIMoUpdate** API takes both the resource path and XML payload as input and performs the HTTP PUT operation on the given resource path.

Note The payload must be in XML format.

The following example shows how to invoke the group@UPDATE XML REST API to update a user group:

```
importClass(javax.xml.parsers.DocumentBuilder);
importClass(javax.xml.parsers.DocumentBuilderFactory);
importClass(javax.xml.parsers.ParserConfigurationException);
importPackage(java.io);
importPackage(javax.xml.transform);
importPackage(javax.xml.transform.dom);
importPackage(javax.xml.transform.stream);
importClass(org.w3c.dom.CDATASection);
importClass(org.w3c.dom.Document);
importClass(org.w3c.dom.Element);
importClass(org.w3c.dom.Node);

logger.addInfo("Demo for userAPIMoUpdate REST API");
logger.addInfo("Execute group@UPDATE HTTP PUT XML REST API through userAPIMoUpdate");

//Update a user group
var resourcePath = "/group";
//Construct payload for group@UPDATE
//parameters to update a user group
var userGroupId = "3";
var grpDescription = "Group description updated";
var grpEmailId = "sdkgroup@abc.com";
var grpCode = "SDKGRP";

function executeUserAPIMoUpdate() {
    var xmlPayload = constructPayload();
    logger.addInfo("XML payload string to update a user group: " + xmlPayload);
    var response = ctxt.getAPI().userAPIMoUpdate(resourcePath, xmlPayload);
```

```
    logger.addInfo("User Group Update response : " + response);
}
function constructPayload() {
    var dbf = DocumentBuilderFactory.newInstance();
    var builder;
    var xmlStr = null;
    builder = dbf.newDocumentBuilder();
    var doc = builder.newDocument();
    var cuicOperationRequest = doc.createElement("cuicOperationRequest");
    var payload = doc.createElement("payload");
    var modifyGroupConfig = doc.createElement("ModifyGroupConfig");
    var groupId = doc.createElement("groupId");
    groupId.appendChild(doc.createTextNode(userGroupId));
    var groupDescription = doc.createElement("groupDescription");
    groupDescription.appendChild(doc.createTextNode(grpDescription));
    var groupContact = doc.createElement("groupContact");
    groupContact.appendChild(doc.createTextNode(grpEmailId));
    var groupCode = doc.createElement("groupCode");
    groupCode.appendChild(doc.createTextNode(grpCode));

    modifyGroupConfig.appendChild(groupId);
    modifyGroupConfig.appendChild(groupDescription);
    modifyGroupConfig.appendChild(groupContact);
    modifyGroupConfig.appendChild(groupCode);

    doc.appendChild(cuicOperationRequest);
    xmlStr = getXMLString(modifyGroupConfig);
    var cdata = doc.createCDATASection(xmlStr);
    payload.appendChild(cdata);
    cuicOperationRequest.appendChild(payload);
    xmlStr = null;
    xmlStr = getXMLString(doc);
    return xmlStr;

}
function getXMLString(node) {
    var xmlStr = null;
    try {
        var domSource = new DOMSource(node);
        var writer = new StringWriter();
        var result = new StreamResult(writer);
        var transformerFactory = TransformerFactory.newInstance();
        var transformer = transformerFactory.newTransformer();
        transformer.transform(domSource, result);
        xmlStr = writer.toString();
    } catch (ex) {
        throw ex;
    }
    return xmlStr;
}
try {
    executeUserAPIMoUpdate();
} catch (e) {
    logger.addError("Error:" + e);
    ctxt.setFailed("Task failed to execute 'userAPIMoUpdate' REST API");
    ctxt.exit();
}
```

userAPIMoDelete

The **userAPIMoDelete** API invokes any one of the HTTP DELETE XML REST APIs from Cisco UCS Director. The **userAPIMoDelete** API takes both the resource path and payload as input, and performs the HTTP DELETE operation on the given resource path.

Note The payload must be in XML format.

The following example shows how to invoke the policyCatalog@DELETE XML REST API to delete a policy catalog:

```
logger.addInfo("Demo for userAPIMoDelete REST API");
logger.addInfo("Execute policyCatalog@DELETE HTTP DELTE XML REST API through
userAPIMoDelete");
//Delete a policy Catalog
var resourcePath = "/policyCatalog";
var catalogName = "advance_cat_1";
var deleteCatalogURL = resourcePath + "/" + catalogName;

//For policyCatalog@DELETE  payload is not required
function executeUserAPIMoDelete() {
    var response = ctxt.getAPI().userAPIMoDelete(deleteCatalogURL, "");
    logger.addInfo("Policy Catalog deletion response : " + response);
}
try {
    executeUserAPIMoDelete();
} catch (e) {
    logger.addError("Error:" + e);
    ctxt.setFailed("Task failed to execute 'userAPIMoDelete' REST API");
    ctxt.exit();
}
```