



## **Cisco UCS Director Cloupiascript Cookbook, Release 6.5**

**First Published:** 2017-07-11

**Last Modified:** 2017-09-11

### **Americas Headquarters**

Cisco Systems, Inc.  
170 West Tasman Drive  
San Jose, CA 95134-1706  
USA  
<http://www.cisco.com>  
Tel: 408 526-4000  
800 553-NETS (6387)  
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <http://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2017 Cisco Systems, Inc. All rights reserved.



## CONTENTS

---

### CHAPTER 1

#### [New and Changed Information for This Release](#) 1

[New and Changed Information for this Release](#) 1

---

### CHAPTER 2

#### [Overview](#) 3

[Cisco UCS Director, Custom Tasks, and CloupiaScript](#) 3

[Structure of an Example](#) 3

[How to Use the Examples](#) 4

[Getting Inputs and Outputs of a Service Request](#) 5

[How to Access XML REST APIs](#) 5

[userAPIMoCreate](#) 5

[userAPIMoQuery](#) 7

[userAPIMoUpdate](#) 8

[userAPIMoDelete](#) 9

---

### CHAPTER 3

#### [Examples](#) 11

[Logging in CloupiaScript](#) 12

[Handling Numeric Input](#) 13

[SSH to a Network Device](#) 14

[Accessing a Network Element Account](#) 15

[Viewing the Virtual and Physical Accounts](#) 16

[Accessing Reports](#) 19

[Accessing a Delegate API](#) 21

[Accessing and Operating on a Database Table](#) 21

[Provisioning a Virtual Machine Multiple Times](#) 22

[Calling or Executing a Workflow Task](#) 25

[Setting a Task to Successful or Failed Status](#) 26

[Invoking Another Workflow](#) 27

[Calling an API on a Remote Cisco UCS Director](#) 28

Executing a Critical Section	29
Obtaining Complete List of VLANs across PODs	30
Obtaining Complete List of IPs Used per Pod	31
Locking or Unlocking a VDC	32
Creating a Task to Obtain the List of Hosts from a VMware Cluster	33
Moving Multiple VMs across VDCs	34
Rolling Back a Task in a Workflow	36
Integrating with ServiceNow Ticketing	36
Sending Emails from the Cloupia Script	39
Archiving Older Service Requests Automatically	40
Determining the Workflow Submitter Information	41
Resizing a Virtual Machine Disk	42
Uploading a Jar File	43
Invoking a Library for a Custom Task	44
Using the Registered List of Values (LOV) in a Custom Task	45
Using a Tabular Report in a Custom Workflow Task	46



CHAPTER

1

# New and Changed Information for This Release

- [New and Changed Information for this Release, page 1](#)

## New and Changed Information for this Release

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 and Changed Behavior in Cisco UCS Director, Release 6.5**

Feature	Description	Where documented
Access XML REST API	Provides a list of APIs to invoke XML REST API to perform basic CRUD operations.	<a href="#">How to Access XML REST APIs, on page 5</a>
Cloupiascript examples	Updated the following examples: <ul style="list-style-type: none"> <li>• Provisioning a Virtual Machine Multiple Times</li> <li>• Moving Multiple VMs across VDCs</li> <li>• Viewing the Virtual and Physical Accounts</li> <li>• Invoking Another Workflow</li> </ul>	<ul style="list-style-type: none"> <li>• <a href="#">Provisioning a Virtual Machine Multiple Times, on page 22</a></li> <li>• <a href="#">Moving Multiple VMs across VDCs, on page 34</a></li> <li>• <a href="#">Viewing the Virtual and Physical Accounts, on page 16</a></li> <li>• <a href="#">Invoking Another Workflow, on page 27</a></li> </ul>





## Overview

---

This chapter contains the following sections:

- [Cisco UCS Director, Custom Tasks, and CloupiaScript, page 3](#)
- [Structure of an Example, page 3](#)
- [How to Use the Examples, page 4](#)
- [Getting Inputs and Outputs of a Service Request, page 5](#)
- [How to Access XML REST APIs, page 5](#)

## 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 CloupiasScript, 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 CloupiasScript and, along with the CloupiasScript 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 workflow 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);
        }
    } catch (e) {
        logger.addError(e);
    }
}

```

```

        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 DELETE 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();
}
```



## Examples

---

This chapter contains the following sections:

- [Logging in CloupiaScript, page 12](#)
- [Handling Numeric Input, page 13](#)
- [SSH to a Network Device, page 14](#)
- [Accessing a Network Element Account, page 15](#)
- [Viewing the Virtual and Physical Accounts, page 16](#)
- [Accessing Reports, page 19](#)
- [Accessing a Delegate API, page 21](#)
- [Accessing and Operating on a Database Table, page 21](#)
- [Provisioning a Virtual Machine Multiple Times, page 22](#)
- [Calling or Executing a Workflow Task, page 25](#)
- [Setting a Task to Successful or Failed Status, page 26](#)
- [Invoking Another Workflow, page 27](#)
- [Calling an API on a Remote Cisco UCS Director, page 28](#)
- [Executing a Critical Section, page 29](#)
- [Obtaining Complete List of VLANs across PODs, page 30](#)
- [Obtaining Complete List of IPs Used per Pod, page 31](#)
- [Locking or Unlocking a VDC, page 32](#)
- [Creating a Task to Obtain the List of Hosts from a VMware Cluster, page 33](#)
- [Moving Multiple VMs across VDCs, page 34](#)
- [Rolling Back a Task in a Workflow, page 36](#)
- [Integrating with ServiceNow Ticketing, page 36](#)
- [Sending Emails from the Cloupia Script, page 39](#)

- [Archiving Older Service Requests Automatically](#), page 40
- [Determining the Workflow Submitter Information](#), page 41
- [Resizing a Virtual Machine Disk](#), page 42
- [Uploading a Jar File](#), page 43
- [Invoking a Library for a Custom Task](#), page 44
- [Using the Registered List of Values \(LOV\) in a Custom Task](#), page 45
- [Using a Tabular Report in a Custom Workflow Task](#), page 46

## Logging in CloupiasScript

### Objective

Print messages to the Service Request log.

### Context

You want to view the output log statements.

### Prerequisites

None

### Components

The Logger object provides the following methods for logging:

- `addDebug` – Displays the debug messages in gray.
- `addInfo` – Displays the normal messages in black.
- `addWarning` – Displays the warning messages in orange.
- `addError` – Displays the error messages in red.

### Code

```
logger.addDebug("About to process the user request");  
logger.addInfo("User "+ctxt.getUserId()+" has requested to provision this");  
logger.addWarning("Resource has reached maximum capacity.");  
logger.addError("Failed to provision the resource.");
```

### Results

The messages that are passed to the logger object are printed.

### Implementation

No modifications required to implement this code.



# Handling Numeric Input

## Objective

Process a numeric input in a custom task.

## Prerequisites

The custom task must first be defined in Cisco UCS Director with a text or numeric input.

## Components

Custom task libraries are not required.

## Code

```
// Handle a number defined as a text input
function handleNumbers() {
  var strInput = input.stringInput;
  var convertedCustomTaskStringInput = null;
  if(strInput != null) {
    // Use the Java Integer wrapper object
    convertedCustomTaskStringInput = new java.lang.Integer(strInput);
    logger.addInfo("convertedCustomTaskStringInput = " +
      convertedCustomTaskStringInput);
  }
}
handleNumbers();

// Handle a number defined as a numeric input
function handleNumbers() {
  var strInput = input.Input * 2;
  var convertedCustomTaskStringInput = null;
  if(strInput != null) {
    // Use the java Integer wrapper object
    convertedCustomTaskStringInput = strInput;
    logger.addInfo("convertedCustomTaskStringInput = " +
      convertedCustomTaskStringInput);
  }
}
handleNumbers();
```

## Results

The script prints the input value to the log.

## Implementation

The implementation is straightforward: take the input variable from the custom task's built-in input object and provide it as an argument to the Java Integer wrapper object. If the given input is not a number, a `NumberFormatException` error is thrown that appears in the log.

# SSH to a Network Device

## Objective

Access a network device using SSH.

## Context

You want a custom task to log into a device (for example, a VM) using SSH and run a command on the device.

## Prerequisites

An SSH-enabled device should be available on the network with a known IP address, username, and password. The custom task is defined with the following inputs:

- ipAddress (IP address of the remote device)
- username (on the device)
- password (on the device)
- command (a command to issue to the device)

## Components

com.cloupia.lib.util.ssh.SSHClient - This API connects to a remote ssh server or device and executes a command, printing the result to standard out (stdout).

## Code

```
importPackage(com.cloupia.lib.util.ssh);
importPackage(java.io);

function testSSHClient() {
    var client = new SSHClient(input.ipAddress, 22, // 22 = SSH standard port
input.userName, input.password);
    client.connect();
    var session = client.openShell(511,25); // (511, 25) = (columns, rows)
    var shellStream = new PrintStream(session.getOutputStream());
    shellStream.println(input.command);
    shellStream.flush();
    client.disconnect();
}

testSSHClient();
```

## Results

An SSH connection is opened and the command is sent to the remote device. The connection is then closed.

### Implementation

The `client.openShell()` method in the example code internally uses a thirdparty library API: `com.maverick.ssh.SshSession.requestPseudoTerminal()`.

This script does not verify the result of the command. In fact, it does not open the session input stream at all, and simply disconnects after sending the command. You would obviously want to do more error checking if you implemented this script in production.

## Accessing a Network Element Account

### Objective

Connect the network device and send CLI commands.

### Context

You want to send the CLI command to the device.

### Prerequisites

The network device should have been added in UCS Director.

### Components

- `NetworkDeviceManager` - This component accepts device details as an argument and establishes a connection to the device. After the connection is created, the user can manage the network device by sending the commands.
- `com.cloupia.lib.cIaaS.network.model.DeviceCredential` - The 'DeviceCredential' API holds device details.

### Code

```
importPackage(com.cloupia.feature.networkController);
importPackage(com.cloupia.feature.networkController.model);
importPackage(com.cloupia.lib.cIaaS.network.model);
importPackage(com.cloupia.feature.networkController.collector);
importPackage(com.cloupia.lib.util);

var devCreds = NetworkPersistenceUtil.getDeviceCredential(dcName, devIP);
var status = NetworkPersistenceUtil.getDeviceStatus(dcName, devIP);
var device = NetworkDeviceManager.getDevice(devCreds);
var failedMessages = new ArrayList();
var cmdAndOutputMsgs = new ArrayList();
var errCounter = new Counter();
var script = new CLIScript();
script.addLine("<cli command here>");
script.execute(device, errCounter, failedMessages, cmdAndOutputMsgs);

// Log commands and their responses
NetworkDBUtil.logCommandsAndResponses(actionLogger, devCreds, cmdAndOutputMsgs);

// Append any exceptions to action logger
NetworkDBUtil.logCommandExceptions(actionLogger, devCreds, errCounter,
failedMessages);
```

**Results**

The command is executed on the device and the command output is printed in the log.

**Implementation**

No modifications required to implement this code.

## Viewing the Virtual and Physical Accounts

**Objective**

View the virtual and physical account details.

**Context**

You want to view the name and configuration of virtual and physical account before executing any action on the account.

**Prerequisites**

The virtual or physical account should be added in Cisco UCS Director.

**Components**

netapp:userAPIGetNetAppAccounts API—This API obtains details about the NetApp device.

ucsm:userAPIGetUCSMAccounts—This API obtains details about the Cisco UCS Manager.

**Code**

```
loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
```

For NetApp accounts:

```
function netappAccounts(){
var opName = "netapp:userAPIGetNetAppAccounts";
var payload = {};
var jsonString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName, jsonString);
logger.addInfo(result);
var resultObj = JSON2.parse(result);
or(var i=0;i<resultObj.serviceResult.length;i++){
logger.addInfo(resultObj.serviceResult[i].accountName);
}
}
netappAccounts();
```

For Cisco UCS Manager account:

```
function ucsmAccounts(){
var opName = "ucsm:userAPIGetUCSMAccounts";
var payload = {};
var jsonString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName, jsonString);
logger.addInfo(result);
var resultObj = JSON2.parse(result);
for(var i=0;i<resultObj.serviceResult.length;i++){
logger.addInfo(resultObj.serviceResult[i].accountName);
}
}
ucsmAccounts();
```

Get the account type by passing the account name:

```
function vmAccounts(){
var opName = "accounts:userAPIGetAccountTypeByName";
var payload = {};
payload.param0 = "v70";
var jsonString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName, jsonString);
logger.addInfo(result);
var resultObj = JSON2.parse(result);
logger.addInfo(resultObj.serviceResult);
}
vmAccounts();
```

Execute the VMware VM operations using the userAPIExecuteVMAction API:

```
function vmActions(){
var opName = "userAPIExecuteVMAction";
var payload = {};
payload.param0 = 46;
payload.param1 = "powerOn";
payload.param2 = "powerOn through executeVmAction";
var jsonString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName, jsonString);
logger.addInfo(result);
var resultObj = JSON2.parse(result);
logger.addInfo(resultObj.serviceResult);
}
vmActions();
```

Read the HyperV SCVMM identities:

```
function getSCVMM(){
var opName = "/hypervSCVMMCloudIdentity";
var result = ctxt.getAPI().getMoResourceAsJson(opName);
logger.addInfo(result);
}
getSCVMM();
```

Read all the physical accounts:

```
function readPhysicalAccounts(){
var opName = "/account";
var result = ctxt.getAPI().getMoResourceAsJson(opName);
logger.addInfo(result);
}
```

```
readPhysicalAccounts();  
Read all the virtual accounts (cloud):  
  
function readVirtualAccounts() {  
  var opName = "/cloud";  
  var result = ctxt.getAPI().getMoResourceAsJson(opName);  
  logger.addInfo(result);  
}  
readVirtualAccounts();
```

### Results

The virtual and physical account details are displayed.

### Implementation

The functions to read the account details are defined in Cisco UCS Director. If required, they can be reused by accessing the feature API instead of redefining the same functions.

## Accessing Reports

### Objective

Access the existing Cisco UCS Director reports and filter the report data.

### Context

You want to obtain a report and filter the data in the report per requirements.

### Prerequisites

Report Name and Context are required to fetch the reports.

### Components

`ctxt.getAPI().getConfigTableReport` - This method returns the `TabularReport` object, which contains the report details.

`ctxt.getAPI().getTabularReport` - This method returns the `TabularReport` object, which contains the report details.

**Code**

```

importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.lib.util.managedreports);

function getReport(reportContext, reportName)
{
    var report = null;
    try
    {
        report = ctxt.getAPI().getConfigTableReport(reportContext, reportName);
    } catch(e)
    {
    }
    if (report == null)
    {
        return ctxt.getAPI().getTabularReport(reportName, reportContext);
    } else
    {
        var source = report.getSourceReport();
        return ctxt.getAPI().getTabularReport (source, reportContext);
    }
}

function getReportView(reportContext, reportName)
{
    var report = getReport(reportContext, reportName);

    if (report == null)
    {
        logger.addError("No such report exists for the specified context
        "+reportName);
        return null;
    }
    return new TableView(report);
}

function accessReports(){
//Refer the following code to create report context.
var reportName="TABULAR_REPORT_PER_CLOUD_VMS_CONFIG_REPORT";
//To create a context, pass the values for arguments in the given order: type,
cloudName, APIReportID. To get these values, click Report Metadata in the corresponding
report.
var repContext = new ReportContext( 1, "vmware109","VMS-T0" );
logger.addInfo("reportContext "+repContext);
var report = getReportView(repContext, reportName);
logger.addInfo("Report:"+report);
//The report can be filtered by columns.
//To retrieve the VMs in the ON power state, use the following line:
report = report.filterRowsByColumn("Power State", "ON", false);
var rowCnt = report.rowCount();
logger.addInfo("Number of powered on Vms: "+ rowCnt );
}
accessReports();

```

**Results**

The code fetches the VM report and filters the data according to the 'ON' Power State.

**Implementation**

No modifications required to implement this code.



# Accessing a Delegate API

## Objective

Access the delegate API by passing the namespace of delegate in the `accessDelegateAPI` method of `APIProvider.java`.

## Context

You want to access delegate APIs using namespace and perform further operations.

## Prerequisites

Namespace is required to access delegate API implementation instance.

## Code

```
function accessDelegateAPI () {  
    var api = ctxt.getAPI ();  
    var delegateObj = api.getAPIDelegate ("chargeback:userAPICheckFunds");  
}  
accessDelegateAPI ();
```

## Results

The above sample returns the delegate API implementation instance (`delegateObj`). You can execute any method in the delegate class.

## Implementation

No modifications required to implement this code.

# Accessing and Operating on a Database Table

## Objective

Access and operate a table of a Cisco UCS Director database by passing the corresponding PoJo. (PoJo represents a database table.)

## Context

You want a script to access a database table and obtain its details.

## Prerequisites

Requires a valid PoJo class name and a valid query to retrieve information.

## Components

`ObjStoreHelper.getStore` - This method accesses the database table and obtains its details.

### Code

```
var accountName = "UCSM_98";
var spDn = "org-root/org-DEV_Org/ls-finalTemp2";
var query = "accountName == '" + accountName +
    "'" && serviceProfileName == '" + spDn + "'";
var store = ObjStoreHelper.getStore(new UcsServer().getClass());
var server = store.query(query);
logger.addInfo("SIZE:"+server.size());
```

### Results

The `getStore()` method of the `ObjStoreHelper` class takes the name of a model class as input and returns the corresponding object store as output.

### Implementation

No modifications required to implement this code.

## Provisioning a Virtual Machine Multiple Times

### Objective

Provision a virtual machine (VM) multiple times.

### Context

To provision a VM for  $n$  number of times in a virtual data center (VDC).

### Prerequisites

Enter the following details to provision a VM multiple times:

- `vDcName`—Name of a VDC in which the VM has to be provisioned.
- `catalogName`—Name of the catalog used for provisioning a VM.
- `qty`—Number of times the VM has to be provisioned.
- `waitTime`—Time to wait in-between the VM provisioning.

### Components

- `provisionVM(vDcName, catalogName, qty, waitTime)` function—Execute this function to provision a VM for  $n$  number of times on a VDC.
- `userAPISubmitServiceRequest` API—Submit a service request to create a VM.

**Code**

```
importPackage(java.lang);
loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
```

From the input, you need to load the following objects:

```
var vDcName = input.selectvDc; //VDC name
var catalogName = input.selectCatalog; //Catalog name
var qty = input.times; //Number of times you want to provision a VM
var waitTime = input.waitTime; //Time to wait in-between the VM provisioning
```

Set an array to store array of service request ID (SR ID) fetched from the API response:

```
var childSrIdArray = [];
```

Define the logger methods for logging messages:

```
logger.addInfo("Times for provision : " + qty);
logger.addInfo("Wait Time : " +waitTime);
logger.addInfo("vDC : " + vDcName);
logger.addInfo("Catalog : " + catalogName);
```

Define the addError method to display an error message when the number of times for VM provisioning is set as zero:

```
if(qty == 0){
logger.addError("Error : Please provide the valid quantity of vm for provisioning.");
ctxt.setFailed("Unable to Provision catalog");
ctxt.exit();
}
```

Provision a VM multiple times using the provisionVM () function:

```
function provisionVM(vDcName, catalogName, qty, waitTime)
{
```

Create a VM using the userAPISubmitServiceRequest API:

```
var opName = "userAPISubmitServiceRequest";
```

Comment is passed as an input parameter to call method:

```
var comment = "";
```

Set the duration of the VMs as -1 (or number of hours):

```
var duration = -1;
```

Set the begin time:

```
var beginTime = -1;
```

Create the payload for the userAPISubmitServiceRequest API:

```
var payload = {};
```

```
payload.param0 = catalogName;
payload.param1 = vDcName;
payload.param2 = duration;
payload.param3 = beginTime;
payload.param4 = 1;
payload.param5 = comment;
```

```
var strPayload = JSON2.stringify(payload);
```

Create a loop to provision the VM *n* number of times:

```
for (var ctr = 0; ctr < qty; ctr = ctr + 1)
{
logger.addInfo("Provision VM =" + (ctr+1) + " of "+qty);
var apiRes = ctxt.getAPI().performOperationOnJSONPayload(opName, strPayload);
if (apiRes == null){
throw "Unable to Provision VM";
}
var parseResponse = JSON2.parse(apiRes);
if(parseResponse.serviceError != null){
logger.addError("Error : " + parseResponse.serviceError);
ctxt.setFailed("Unable to Provision VM");
ctxt.exit();
}
}
```

Use the thread sleep method to delay between provisioning:

```
var srId = parseResponse.serviceResult;
childSrIdArray[ctr] = srId;
```

```
var milliseconds = waitTime * 1000;
Thread.sleep(milliseconds);
}
```

Create a loop through array of SR ID to get the SR status (success or failed):

```
for (var i=0; i<childSrIdArray.length; i++)
{
var childSrId = childSrIdArray[i];
var status = ctxt.waitForCompletion(childSrId, 1800000);
if (status == 0)
{
logger.addInfo("Provisioned SR ID =" + childSrId+ " successfully.");
} else {
logger.addError("SR ID =" + childSrId+ " failed");
}
}
}
```

Call the provisionVM function using the try and catch statement:

```
try {
provisionVM(vDcName, catalogName, qty, waitTime);
}

catch (e) {
logger.addError("Error: " + e);
ctxt.setFailed("Unable to Provision VM");
ctxt.exit();
}
```

### Results

The VM is provisioned on the VDC for *n* number of times as defined in the *qty* variable.

### Implementation

See the code above for modifications you might require to implement the code.

## Calling or Executing a Workflow Task

### Objective

Invoke a workflow task from Cloupiascript using an inner task.

### Prerequisites

Inner task name and set of inputs are required.

### Components

ctxt.createInnerTaskContext() - Use this method to access the task variables and to set the input values.

### Code

```
// Task Label: Create VM Disk
// Task Name: Create VM Disk

function Create_VM_Disk()
{
    var task = ctxt.createInnerTaskContext("Create VM Disk");

    // Input 'Select VM', mandatory=true, mappableTo=vm
    task.setInput("Select VM", input.vmId);

    // Input 'Disk Size (GB)', mandatory=true, mappableTo=gen_text_input
    task.setInput("Disk Size (GB)", input.diskSize);

    // Input 'Select Disk Type', mandatory=true, mappableTo=gen_text_input
    task.setInput("Select Disk Type", input.diskType);

    // Input 'Select Datastore', mandatory=false, mappableTo=dataStoreName
    task.setInput("Select Datastore", input.datastoreName);

    // Input 'Thin Provisioning', mandatory=false, mappableTo=
    task.setInput("Thin Provisioning", input.thinProvision);

    // Execute the task. On failure, the task throws an exception error
    task.execute();
}

// Invoke the task
Create_VM_Disk();
```

### Results

The script executes the Create VM task as inner task.

### Implementation

No modifications required to implement this code.

## Setting a Task to Successful or Failed Status

### Objective

Set the Task status to Successful or Failed based on the given condition.

### Prerequisites

None

### Components

ctxt - Used to set the status message of the workflow task.

**Code**

```
if (true) {
    ctxt.setFailed("Error output message"); // to set the task status as failed
    ctxt.exit(); // to exit task flow normally
} else {
    ctxt.setSuccessful(); // to set the task status to successful
}
```

**Results**

Sets the task status as failed or successful based on the given condition.

**Implementation**

No modifications required to implement this code.

## Invoking Another Workflow

**Objective**

Invoke a workflow from another workflow.

**Context**

You want to invoke another workflow using CloupiaScript.

**Prerequisites**

Workflow name and workflow parameters are required.

**Components**

Using the `userAPISubmitWorkflowServiceRequest("workflow_name", params, parent_srId)` API, you can invoke another workflow. The following parameters are required:

- `workflow_name`: Name of the workflow to be invoked.
- `params`: List of parameters to be passed to the invoking workflow.
- `parent_srId`: Links the child workflow to the service request ID.

### Code

```

loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
function submitWorkflow(){
var opName = "userAPISubmitWorkflowServiceRequest";
var payload = {};
payload.param0 = "wfname";
//You can give inputs through param1 parameter in the name and value JSON format.
var inputSet1 = {};
inputSet1.name = "inputname1";
inputSet1.value = "value1";
var inputSet2 = {};
inputSet2.name = "inputname2";
inputSet2.value = "value2";
var inputList = new Array();
inputList.push(inputSet1);
inputList.push(inputSet2);
var listObj = {};
listObj.list = inputList;
payload.param1 = listObj;
payload.param2 = -1;
var jsonInString = JSON2.stringify(payload);
logger.addInfo(jsonInString);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName, jsonInString);
logger.addInfo(result);
var resultObj = JSON2.parse(result);
if (resultObj.serviceError != null) {
    logger.addError(resultObj.serviceError);
    ctxt.setFailed("Task Failed to submit workflow");
    ctxt.exit();
}
logger.addInfo("SR Id: "+resultObj.serviceResult);
output.OUTPUT_SRID = resultObj.serviceResult;
}
submitWorkflow();

```

### Results

The above script invokes another workflow with the given inputs and generates a service request.

### Implementation

The inputname1 and inputname2 provided in the parameter list must be the same as the Input\_Name defined in the workflow that is being invoked. The parameter parent\_srId is required to make the invoking workflow as child.

The child service request won't be visible in the Service Request logs. If you want to invoke the workflow as separate service request, pass parent\_srId as -1.

## Calling an API on a Remote Cisco UCS Director

### Objective

Execute an API remotely by selecting the required server IP address in the API browser.

### Context

You want to call an API on a remote Cisco UCS Director.



### Prerequisites

Requires a remote UCSD server.

### Components

JsonUtil - Sends the RestAPI call remotely.

CloupiaClient - Creates the connection.

### Code

```
var client = new CloupiaClient(server, 443, key);
// first parameter is the remote server-address, the second parameter is remote
//server-port, while the third parameter is rest-api-access-key.
JsonUtil.prettyPrint(client.sendJSONRequest(opName, paramList));
// first parameter is operationName / or API name, second parameter is the API argument
  list as specified in the API signature.
```

To obtain a tabular report:

```
client.sendJSONRequest("userAPIGetTabularReport", Arrays.asList(new String[]{"vm",
"2319", "DISKS-T0"}));
```

To obtain all catalogs:

```
client.sendJSONRequest("userAPIGetAllCatalogs", null);
```

To obtain all VM actions:

```
client.sendJSONRequest("userAPIGetAvailableVMActions", Arrays.asList(new String[]
{"2293"}));
```

To submit a service request:

```
client.sendJSONRequest("userAPISubmitServiceRequest" , Arrays.asList(new String[]
{cata-logName, vdcName,
durationHours, beginTime, qty, comments }));
```

To execute a workflow :

```
client.sendJSONRequest("userAPISubmitWorkflowServiceRequest", Arrays.asList(new Object[]
{workflowName, nameValueList, -1 }));
// param 1 is workflowName
```

### Results

The script calls the API from the remote UCS Director.

### Implementation

Provide the server IP address and port number to the CloupiaClient method.

## Executing a Critical Section

### Objective

Allow exclusive access to a thread to execute a critical section using the CriticalSection API.

**Context**

The CriticalSection API helps a workflow thread (running with a context sId) gain exclusive access to a critical section with a specified sectionName or lockName. Any other thread wanting access to the critical section must wait until the exitCriticalSection API releases the section from the thread holding the lock.

**Prerequisites**

None

**Components**

CriticalSectionUtil - use the class to guard the critical section.

**Code**

To enter a critical section and effectively obtain a lock, use:

```
public static void enterCriticalSection(CustomActionTriggerContext context,
CustomAction-
Logger logger, String lockName) throws Exception
```

When the thread wants to release all locks held by the service request, use the following API:

```
public static void exitCriticalSection(CustomActionTriggerContext context, CustomAction-
Logger logger) throws Exception
```

**Results**

When there are multiple requests, the above mentioned APIs help execute requests, one at a time.

**Implementation**

No modifications required to implement this code.

## Obtaining Complete List of VLANs across PODs

**Objective**

Obtain all VLAN IDs available within an account.

**Context**

You want to obtain all the VLAN IDs that are configured on the UCS device.

**Prerequisites**

You require access to a UCSM account.

**Components**

UcsDataPersistenceUtil - This class fetches the VLAN IDs.

**Code**

```
importPackage(com.cloupia.feature.ucsController); //UcsDataPersistenceUtil
//importPackage(java.lang);
importPackage(java.util);
importPackage(com.cloupia.lib.cIaaS.ucs.model); //UcsVLAN

//var accountName = input.name;
var accountName = "DR_UCSM30";
var vlanIdsPerAccount = "";
var ucsVlanList = UcsDataPersistenceUtil.getVLAN(accountName);
var vLanList = new ArrayList();
for (var i=0;i<ucsVlanList.size();i++){
var ucsVLAN = ucsVlanList.get(i);
vlanIdsPerAccount = vlanIdsPerAccount+ucsVLAN.getId()+",";
vLanList.add(ucsVLAN.getId());
}
logger.addInfo("No of vlan ids:"+ucsVlanList.size());
vlanIdsPerAccount = vlanIdsPerAccount.substring(0, vlanIdsPerAccount.length()-1);
logger.addInfo(accountName+" "+vlanIdsPerAccount);
//output.vlanIdsPerAccount = vlanIdsPerAccount;
```

**Results**

Comma separated VLAN IDs. For example: DR\_UCSM30:1,1,205,1135,1136,1001.

**Implementation**

No modifications required to implement this code.

## Obtaining Complete List of IPs Used per Pod

**Objective**

Obtain all the virtual account IPs available in one pod.

**Context**

You know a pod name and want to obtain all the Virtual Account IPs from the pod.

**Prerequisites**

Virtual account should be available in the pod.

**Components**

InfraPersistenceUtil.getAllFlexPodAccounts - Returns all the account names added in the pod.

**Code**

```

importPackage(com.cloupia.model.cIM); //Account
importPackage(com.cloupia.service.cIM.inframgr); //InfraPersistenceUtil

var podName = "Default Pod";
// var podName = input.podName;
var allAcct = ctxt.getAPI().getAllAccounts();
logger.addInfo("No of accounts:"+allAcct.length);

var accList = InfraPersistenceUtil.getAllFlexPodAccounts(podName);
logger.addInfo("No of accounts in the pod "+podName+" is:"+accList.size());
var listOfIPsPerPod = "";
for(var count = 0;count < accList.size();count++){
    var acc = accList.get(count);
    logger.addInfo("Account name:"+acc.getAccountName());
    if(acc.getAccountType()==CloudTypes.VMWARE){
        listOfIPsPerPod = listOfIPsPerPod+acc.getVServer()+",";
    } else if(acc.getAccountType()==CloudTypes.HYPERV){
        listOfIPsPerPod = listOfIPsPerPod+acc.getHServer()+",";
    }
}
listOfIPsPerPod = listOfIPsPerPod.substring(0, listOfIPsPerPod.length()-1);
logger.addInfo(podName+" "+listOfIPsPerPod);
output.listOfIPsPerPod = listOfIPsPerPod;
getListOfIPs();

```

**Results**

The output is a comma separated list of all the IPs in the pod. For example: 172.29.110.196, 172.25.168.80, 172.29.109.82.

**Implementation**

No modifications required to implement this code.

## Locking or Unlocking a VDC

**Objective**

To lock or unlock a VDC.

**Prerequisites**

VDC must be created.

**Components**

VDCUtil - Use the method to lock the VDC

VDCUtil.setLocked() - Use the method to lock or unlock VDC

### Code

```
importPackage (com.cloupia.model.cIM);
importPackage (com.cloupia.service.cIM.inframgr);
importPackage (java.util);
function lock()
{
    var flag=false;
    vdcId=input.VDC;
    var vdc;
    try {
        vdc=VDCUtil.getVDC (input.VDC);
        vdc.setLocked(input.VdcLocked);//Pass the value 'true' to lock the VDC.
                                                //To unlock the VDC, pass the value as
false.
        flag=VDCUtil.modifyVDC (vdc);
    }
    catch (e) {
        logger.addError("Exception error when modifying VDC.");
        ctxt.setFailed(e);
        ctxt.exit();
    }
    if(flag){
        logger.addInfo("Successfully modified VDC." );
        vdc=VDCUtil.getVDC (vdcId);
        output.vdcId=vdcId;
        output.isLocked=vdc.isLocked();
    } else {
        logger.addInfo("Unable to modify VDC." );
    }
}

lock();
```

### Results

The script locks the VDC. This can be verified in UCS Director.

### Implementation

No modifications required to implement this code.

## Creating a Task to Obtain the List of Hosts from a VMware Cluster

### Objective

Obtain the list of hosts from a VMware Cluster.

### Context

You want to view all the hosts in a VMware cluster.

### Prerequisites

A VMware cluster must be available.

### Components

InfraPersistenceUtil - This object obtains the list of hosts from a VMware cluster.

### Code

```
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(java.util);

function getListOfHosts(){
var listOfHosts =
InfraPersistenceUtil.getVMWareHostsByCluster(input.VMWAREACCOUNT,input.CLUSTER);
return listOfHosts;
}
```

### Results

Comma separated hosts are mapped to the Associate VNX LUN as a datastore task. However, this operation is applicable on all hosts in a cluster, not just a single host.

### Implementation

No modifications required to implement this code.

## Moving Multiple VMs across VDCs

### Objective

Move all VMs from one VDC to another VDC.

### Prerequisites

VDC must be available to move the VMs.

### Components

- createMoResource(moveVmsResourcePath, requestPayloadStr)—Execute this method to move the VMs to the VDC.

**Code**

```

loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
Set the resource path to move VMs to VDC:

var moveVmsResourcePath = "/vmwareVM";
Get the account name, VM ID, and VDC name from user:

var accountName = input.accountName;
var vmId = input.vmId;
var vdcName = input.vdcName;
Define the logger methods for logging messages:

logger.addInfo("Cloud Name : "+input.accountName);
logger.addInfo("VM/s Id : "+input.vmId);
logger.addInfo("vDC Name : "+input.vdcName);
Create a request to move VMs to VDC using the moveVmsTovDC function:

function moveVmsTovDC(accountName, vmId, vdcName)
{
var moveVmsTovDCObj = {};
moveVmsTovDCObj.accountName = accountName;
moveVmsTovDCObj.vmId = vmId;
moveVmsTovDCObj.vdcName = vdcName;
var requestObj = {};
requestObj.AssignVMstoVDC = moveVmsTovDCObj;
var req = {};
req.operationType = "MOVE_VMS_TO_VDC";
req.payload = requestObj;
var requestPayload = {};
requestPayload.cuicOperationRequest = req;
Creating the payload request:

var requestPayloadStr = JSON2.stringify(requestPayload);
Execute the payload using the createMoResource method:

var res = ctxt.getAPI().createMoResource(moveVmsResourcePath, requestPayloadStr);
logger.addInfo("Response : " + res);
var resObj = JSON2.parse(res);
Check the operation status of errorMessage:

var opStatus = resObj.cuicOperationResponse.operationStatus;
if( opStatus != 0){
var errorMessage = resObj.cuicOperationResponse.errorMessage;
logger.addError(errorMessage);
ctxt.setFailed(errorMessage);
ctxt.exit();
}
}
Call the moveVmsTovDC function using the try and catch statement to handle exceptions:

try {
moveVmsTovDC(accountName, vmId, vdcName);
}
catch (e) {
logger.addError("Error: " + e);
ctxt.setFailed("Unable to MOVE VM/s TO VDC");
ctxt.exit();
}

```

**Results**

The selected VMs in a VDC are moved to another VDC.

**Implementation**

See the code above for modifications you might require to implement the code.

# Rolling Back a Task in a Workflow

## Objective

Roll back a task in a workflow using the change tracker API.

## Context

You performed an operation and you now want to revert it. For example, you created a VM, but then want to roll back and delete the VM.

## Prerequisites

The workflow should have been successfully executed.

## Components

The undo task handler name and undo config object are the important parameters to execute the undo task. The undo task needs the data for the original task to undo the actions done by the original task.

Related API for undoing modify/delete operations: `ChangeTracker.undoableResourceModified` and `ChangeTracker.undoableResourceDeleted`.

Both the API calls have the same set of parameters. Each task in the workflow needs undo support using the change tracker API to successfully roll back the entire workflow.

## Code

```
importPackage(com.cloupia.service.cim.inframgr.customactions);
importPackage(com.cloupia.feature.accounts.wftasks);
function doRollBack(){
    var undoTaskHandlerName = DeleteGroupConfig.HANDLER_NAME;
    var configObject = new DeleteGroupConfig(input.groupId+ "")
    ChangeTracker.undoableResourceModified(input.assetType, input.assetId, input.assetLabel,
    input.description, undoTaskHandlerName, configObject) ;
}
doRollBack();
```

## Results

The group with the given ID is deleted after executing the task.

## Implementation

No modifications required to implement this code.

# Integrating with ServiceNow Ticketing

## Objective

Reset data on a demo instance using ServiceNow.



**Context**

ServiceNow is a software platform that supports IT service management and automates common business processes. This software as a service (SaaS) platform contains a number of modular applications that can vary by instance and user. It uses the HttpClient API to send the HTTP request and handle the HTTP response.

ServiceNow provides several demo instances, which you can use for testing. The demo login and password is admin/admin. The demo login and password should be verified in ServiceNow, because they change regularly.

**Prerequisites**

ServiceNow resets the data on each demo instance daily. You must enable JSON plugin on ServiceNow to make this API work. Refer to the Service Now Website on how to activate the Plugin section.

**Components**

HttpClient - Used to communicate with the ServiceNow software.

## Code

```

importPackage(java.util);
importPackage(java.io);
importPackage(com.cloupia.lib.util);
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(org.apache.commons.httpclient);
importPackage(org.apache.commons.httpclient.cookie);
importPackage(org.apache.commons.httpclient.methods);
importPackage(org.apache.commons.httpclient.auth);
//var login= ctxt.getInput("LOGIN");
//var password = ctxt.getInput("PASSWORD");
var login = "admin";
var password = "admin";
var instance = "demo011.service-now.com";
var url = "/incident.do?JSON&sysparm_action=insert";
// Link to Service-now documentation
//http://wiki.servicenow.com/index.php?title=JSON_Web_Service#insert
var map = new HashMap();
map.put("sysparm_action", "insert");
map.put("short_description", "Sample incident #5");
map.put("impact", "2");
map.put("urgency", "2");
map.put("caller_id", "Joe Z");
map.put("category", "software");
var data = JSON.javaToJsonObject(map, map.getClass());
logger.addInfo("JSON Data = "+data);
var httpClient = new HttpClient();
httpClient.getHostConfiguration().setHost(instance, 443, "https");
httpClient.getParams().setAuthenticationPreemptive(true);
httpClient.getParams().setCookiePolicy("default");
var defaultcreds = new UsernamePasswordCredentials(login, password);
httpClient.getState().setCredentials(new AuthScope(instance, -1, null), defaultcreds);
var httpMethod = new PostMethod(url);
httpMethod.setRequestEntity(new StringRequestEntity(data));
httpMethod.setRequestHeader("Content-Type", "application/json");
httpClient.executeMethod(httpMethod);
var statuscode = httpMethod.getStatusCode();
logger.addInfo("STATUSCODE = "+statuscode);
if (statuscode != 200)
{
logger.addError("Ticket failed to open, with the following code "+statuscode);
if (statuscode == 302)
{
logger.addWarning("Likely cause of failure is that the JSON plugin is not activated
on the Service Now instance. ");
logger.addWarning("Check documentation on how to enable the plugin: "+
"http://wiki.servicenow.com/index.php?title=JSON_Web_Service#insert "+
" (see section 2)");
}
httpMethod.releaseConnection();
// Set this task as failed.
ctxt.setFailed("Unable to open ticket");
} else
{
var reader = new InputStreamReader(httpMethod.getResponseBodyAsStream());
var resp = JSON.getJsonElement(reader, null);
logger.addInfo("Response = "+resp);
var entry = resp.get("records").get(0);
logger.addInfo("Ticket Number "+entry.get("number"));
logger.addInfo("Ticket Sys_id "+entry.get("sys_id"));
httpMethod.releaseConnection();
}
}

```

## Results

The script communicates with the ServiceNow software.

## Implementation

Mention the server address and username/passwords before executing the script.

# Sending Emails from the Cloupia Script

## Objective

Send an email after a specific operation has been completed.

## Context

You want to send an email after a particular operation is completed. For example, you want to send an email after a VM has been provisioned.

## Prerequisites

Ensure that the report values are not empty in the Mail Setup screen. To access the Mail Setup screen, log into Cisco UCS Director, and choose **Administration > System** and click **Mail Setup**.

## Components

MailManager — Used to send email.

## Code

```

importPackage (com.cloupia.model.cIM);
importPackage (com.cloupia.lib.util.mail);
importPackage (com.cloupia.fw.objstore);
function getMailSettings()
{
return ObjStoreHelper.getStore((new MailSettings()).getClass()).getSingleton();
}
// Assume the To Email Address is in the input variable 'Email Address'
var toEmail = [ ctxt.getInput("Email Address") ];
Cisco UCS Director Cloupia Script Configuration Guide, Release 5.0
17
Cloupia Script Samples
Sending Emails from the Cloupia Script
var message = new EmailMessageRequest();
message.setToAddr(toEmail);
// other methods in the message object are
// setToAddr(emailAddress) -- Sets a single email address in To list
// setCcAddr(emailAddress) -- Sets a single email address in Cc list
// setBccAddr(emailAddress) -- Sets a single email address in Bcc list
// setCcAddrs(emailAddressArray) -- Sets an array of email addresses in the CC
//list
// setBccAddrs(emailAddressArray) -- Sets an array of email addresses in BCC
//list
message.setSubject("Test Email");
message.setFromAddress("no-reply@cisco.com");
var body = "<h1>This is a sample Email </h1><br><b>Sample content</b>";
message.setMessageBody(body);
// By default, content type is text or HTML. The following method can be used to modify

//content type
message.setContentType("text/plain");
logger.addInfo("Sending email");
MailManager.sendEmail("Sample Email", getMailSettings(), message);

```

**Results**

An email is sent to the respective email recipient.

**Implementation**

No modifications required to implement this code.

# Archiving Older Service Requests Automatically

**Objective**

Set up the Cloupia script to archive all service requests (SRs) that are older than 30 days.

**Context**

You find that the number of SR IDs are high and want to hide the SR IDs in the UI. You want to execute a workflow that makes sure that only the SRs in the last 30 days are shown in the daily report.

**Prerequisites**

Service request ID should be created.

**Components**

WorkflowManager.getInstance().archive() - This method is used to archive SR IDs.

**Code**

```
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.fw.objstore);
importPackage(com.cloupia.service.cIM.inframgr.workflowmgr);
importPackage(com.cloupia.service.cIM.inframgr.cmdb);
importPackage(java.lang);

function getOlderSRs(ageInDays) {
    var timeStamp = System.currentTimeMillis() - (ageInDays*(24*60*60*1000));
    var store = ObjStoreHelper.getStore((new ServiceRequest()).getClass());
    return store.query("isArchived == false && requestTime < "+timeStamp);
}
var srList = getOlderSRs(30);
logger.addInfo("There are "+srList.size()+" SRs to be archived");
for (var i=0; i<srList.size(); i++) {
    try {
        var sr = srList.get(i);
        logger.addDebug("[ "+i+" ] Archiving SR "+sr.getRequestId());
        // Archive the SR
        WorkflowManager.getInstance().archive(sr.getRequestId());
        // Add an entry into the Change Log
        CMDB.getInstance().change(ctxt.getUserId(), 0, "Request archived by Workflow", sr);
    } catch (e) {
        logger.addError("Error :"+e.message);
    }
}
}
```

**Results**

The SR IDs are archived and not shown in UI after the workflow is executed.

**Implementation**

Need to pass the number of days as argument in the line `getOlderSRs()`. The above example archives all service requests that are older than 30 days.

## Determining the Workflow Submitter Information

**Objective**

Access the user details of the person who submitted the workflow, such as the first name, last name, and email address.

**Context**

You want to know the details of the person that submitted the workflow, such as the User ID, first name, last name, and email ID.

**Prerequisites**

None

**Components**

`APILoginProfile` - The `ctxt.getAPI()` and `userAPIGetMyLoginProfile()` methods return the `APILoginProfile` object, which contains the user details.

The following example uses these workflow level variables to capture and save the information that is retrieved by the script and used by other tasks:

- `SUBMITTER_EMAIL`
- `SUBMITTER_FIRSTNAME`
- `SUBMITTER_LASTNAME`
- `SUBMITTER_GROUPNAME`

**Code**

```

importPackage(java.lang);
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.service.cIM.inframgr);

var userId = ctxt.getUserId();
// Get the current workflow submitter's profile
var userProfile = ctxt.getAPI().userAPIGetMyLoginProfile();
var firstName = userProfile.getFirstName();
var lastName = userProfile.getLastName();
var groupName = userProfile.getGroupName();
var groupId = userProfile.getGroupId();
var role = userProfile.getRole();
var email = userProfile.getEmail();
// Add debug statements to SR log
logger.addDebug("UserId="+userId+", Name="+firstName + " "+ lastName +",
Email="+email+", group="+groupName+", "+"Role="+role);
// Save to workflow variables as necessary
ctxt.updateInput("SUBMITTER_EMAIL", email);
ctxt.updateInput("SUBMITTER_FIRSTNAME", firstName);
ctxt.updateInput("SUBMITTER_LASTNAME", lastName);
ctxt.updateInput("SUBMITTER_GROUPNAME", groupName);

```

**Results**

The example script prints the User ID, name, email, group, and the role of the workflow submitter.

**Implementation**

No modifications required to implement this code.

# Resizing a Virtual Machine Disk

**Objective**

Resize the disk of a VM after it has been provisioned.

**Context****Prerequisites**

The VM should have the disk. The library task 'Resize VM Disk' is available for reference.

**Components**

VMWareVMSummary - The ctxt.getAPI().getVMwareVMInfo(vmid) method returns the object, VMWareVMSummary

ctxt.getAPI().performAction() - Performs VM disk resize operation

vmid - Input variable, which points to the VM that needs to be resized

## Code

```
importPackage(java.lang);
importPackage(java.util);
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.service.cIM.inframgr);
function resizeVmDisk(vmidstr, diskName, sizeInGB)
{
    var vmid = Integer.parseInt(vmidstr);
    // create the context to
    var vmcontext = util.createContext("vm", null, vmidstr);
    // obtain VM details
    var vminfo = ctxt.getAPI().getVMwareVMInfo(vmid);
    var vmname = vminfo.getName();
    var nameparam = new ActionParam("vmName", vmname);
    var sizeparam = new ActionParam("vmSize", sizeInGB);
    var diskparam = new ActionParam("vmDiskLabel", diskName);
    var paramarr = [ nameparam, sizeparam, diskparam ];
    logger.addInfo("About to resize VM "+vmidstr+" name="+vmname);
    var status = ctxt.getAPI().performAction(vmcontext,"diskResize","Resizing VM to
test the script",ctxt.getUserId(),paramarr);
    logger.addInfo("status = "+status);
}
var vmidstr1 = ctxt.getInput("VMID");
resizeVmDisk(vmidstr1, "Hard Disk 1", "10");
```

## Results

The VM disk is resized after the script is executed.

## Implementation

No modifications required to implement this code.

## See Also

Refer to additional in-built library tasks in UCS Director.

# Uploading a Jar File

## Objective

Make available an external jar file for a custom task by uploading the jar file through the Script Module.

## Context

You want to access an external jar file for a custom task.

## Prerequisites

Ensure that a .jar file is registered with the script module in Cisco UCS Director. See the Using Script Modules chapter of *Cisco UCS Director Orchestration Guide*.

## Components

Script module

## Code

Suppose the sample jar file "Calculatedemo.jar" contains the following code:

```
package com.calculate;
public class CalculateDemo {
    public int add(int n1,int n2){
        return n1+n2;
    }
    public int multiply(int n1,int n2){
        return n1*n2;
    }
}
```

The following script helps complete the custom task using the Calculatedemo.jar file.

```
loadJar("Mod1/calculatedemo.jar");
importPackage(com.calculate);
var demo = new CalculateDemo();
var demo1 = new com.calculate.CalculateDemo();
logger.addInfo(demo);
var sum = demo.add(5,6);
logger.addInfo("Sum:"+sum);
logger.addInfo(demo1);
var mul = demo1.multiply(3,4);
logger.addInfo("Multiplication:"+mul);
```

## Results

The uploaded jar file is accessed from the custom task successfully.

## Implementation

No modifications required to implement this code.

# Invoking a Library for a Custom Task

## Objective

Invoke library functions for a custom task.

## Context

You have some reusable code and want to use it across multiple custom tasks. You can add the reusable code in a library and invoke the library from the custom task.

## Prerequisites

Create a library with the reusable code in Cisco UCS Director. See the Using Script Modules chapter of *Cisco UCS Director Orchestration Guide*.

## Components

Script Module



### Code

Suppose the library "lib1" contains the following reusable code:

```
function mul(a,b){
return a*b;
}
function add(a,b){
return a+b;
}
```

Invoke the library using the following script:

```
loadLibrary("Mod1/lib1");
var mul = mul(5,6);
logger.addInfo("The product is:"+mul);
var add = add(5,6);
logger.addInfo("The sum is:"+add);
```

### Results

The library is accessed and the custom task is executed successfully.

### Implementation

No modifications required to implement this code.

## Using the Registered List of Values (LOV) in a Custom Task

### Objective

Access a registered LOV from a custom task.

### Context

### Prerequisites

Register the List of Values (LOV) in Cisco UCS Director. See the Using Script Modules chapter of *Cisco UCS Director Orchestration Guide*.

### Components

Script Module

**Code**

```

{
  var lovProvider = new com.cloupia.service.cIM.inframgr.forms.wizard.LOVProviderIf({
  getLOVs : function(session) {
  //Implement this section based on your requirement
  var SiteList = SitePersistenceUtil.getAllClusterSites();
  var formlovs=[];
  if(SiteList==null){
  return formlovs;
  }
  if(SiteList.size()==0){
  return formlovs;
  }
  var formlov;
  for(var count = 0;count<SiteList.size();count++)
  {
  var clusterSite = SiteList.get(count);
  var siteName = clusterSite.getSiteName();
  formlov = new FormLOVPair(siteName, siteName);
  formlovs[count] = formlov;
  }
  return formlovs;
  //End of implementation for Lovprovider
  }
  });
  return lovProvider;
}

```

**Results**

The script helps access a site list of LOVs from a custom task.

**Implementation**

To access an LOV from the custom task, create a custom task and define a variable. The variable name should be the same as the LOV created in the Script Module.

## Using a Tabular Report in a Custom Workflow Task

**Objective**

Access a registered tabular report from a custom task.

**Prerequisites**

Add a tabular report in Cisco UCS Director, and then modify the table to add column entries. See the Using Script Modules chapter of *Cisco UCS Director Orchestration Guide*.

**Components**

Script Module

## Code

Add the following code to the tabular report.

```
{
Varmodel = new TabularReportInternalModel();
model.addNumberColumn("Site ID", "Site ID");
model.addTextColumn("Site Name", "Site Name");
model.addTextColumn("Description", "Description");
model.completedHeader();
//Obtain values from the database and populate the model object as shown below.
//The model object is generated depending on the Column entries.
//model.addNumberValue(0);
//model.addTextValue("Site Name");
//model.addTextValue("Description");
//model.completedRow();
//Start of your implementation. Implement this section based on your requirements.
Var SiteList = SitePersistenceUtil.getAllClusterSites();
For (count = 0;count<SiteList.size();count++)
{
Var site = SiteList.get(count);
model.addNumberValue(site.getClusterSiteId());
model.addTextValue(site.getSiteName());
model.addTextValue(site.getDescription());
model.addTextValue(site.getContactName());
model.completedRow();
}
//End of your implementation
model.updateReport(report);
}
```

## Results

The tabular report is accessed from the custom task and your requirement is implemented.

## Implementation

To access the tabular report from the custom task, create a custom task and define a variable. The variable name should be the same as the tabular report name created in the Script Module.

