Cisco UCS Director Cloupia Script Cookbook

Release 1.4
Published: July, 2014
Revised: Feb, 2015
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Getting Started</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Logging With CloupiaScript</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Custom Tasks</td>
<td>6</td>
</tr>
<tr>
<td>3.1</td>
<td>Adding a New Custom Task</td>
<td>6</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Defining Custom Task Inputs</td>
<td>6</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Defining Custom Task Outputs</td>
<td>7</td>
</tr>
<tr>
<td>3.1.3</td>
<td>Creating the Custom Task Script</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Exporting a Custom Task</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>Importing a Custom Task</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>Handling Numbers in CloupiaScript</td>
<td>11</td>
</tr>
<tr>
<td>4.1</td>
<td>Defining the Input type in the Custom Task</td>
<td>12</td>
</tr>
<tr>
<td>5</td>
<td>Accessing a Network Device</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>Accessing a Network Element Account</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Accessing Session or API for Virtual and Physical accounts</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Accessing reports</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Emailing Reports</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>Accessing a Delegate API</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Accessing and Operating on a Database Table</td>
<td>21</td>
</tr>
<tr>
<td>12</td>
<td>Provisioning a Catalog Multiple times</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>Getting Inputs or Outputs of a Service Request</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>Calling or Executing a Workflow Task</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Setting a Task to Succes or Failed State</td>
<td>25</td>
</tr>
<tr>
<td>16</td>
<td>Invoking Another Workflow</td>
<td>25</td>
</tr>
<tr>
<td>17</td>
<td>Accessing a Feature Class</td>
<td>26</td>
</tr>
<tr>
<td>18</td>
<td>Calling an API on a Remote Cisco UCS Director</td>
<td>26</td>
</tr>
<tr>
<td>19</td>
<td>Critical Section</td>
<td>27</td>
</tr>
<tr>
<td>20</td>
<td>Getting Complete List of VLANs across PODs</td>
<td>27</td>
</tr>
<tr>
<td>21</td>
<td>Getting Complete List of IPs used per-pod or across</td>
<td>28</td>
</tr>
<tr>
<td>22</td>
<td>Creating a Task to get the List of Hosts from a VMware cluster</td>
<td>29</td>
</tr>
<tr>
<td>23</td>
<td>Moving Multi-VM across vDC's</td>
<td>30</td>
</tr>
<tr>
<td>24</td>
<td>Rolling back a Task in a Workflow</td>
<td>31</td>
</tr>
</tbody>
</table>
1 Getting Started

CloupiaScript is a combination of JavaScript and Cloupia libraries, which you can execute from within the Cloupia server platform.

**Highlights**

- You can access a subset of Cloupia Java APIs and objects from the script.
- CloupiaScript supports all JavaScript syntax.
- CloupiaScript does not support browser-based objects as CloupiaScript contains a server-side scripting.

**Script Context Variables**

Three predefined top-level variables in any CloupiaScript are as follows:

- ctxt
- logger
- util

**Object: ctxt**

This variable points to the Cloupia Workflow Execution context. Using this object, the script can access information about the current workflow, task, input, output and can access the Cloupia APIs.

**Object: logger**

This variable points to Cloupia workflow logger object. All scripts must add
reasonable amount of logging statements, which will be logged into the Service Request log.

**Object: util**

Few built-in utility methods can be accessed using this variable.

### Defining and Executing the CloupiaScript

1. Define a workflow.
2. Find the **Execute CloupiaScript** task from the task library and drag it to the workflow designer.
3. Write CloupiaScript.
4. Click **Execute** to execute CloupiaScript.

#### Sample script to print a message:

```javascript
logger.addInfo("Hello World");
```

The preceding script prints the message ‘Hello World” to the log.

#### Sample script to find sum of two numbers

```javascript
var a = 2;
var b = 5;
var total = a+b;
logger.addInfo("Total is " + total);
```

### 2 Logging With CloupiaScript

CloupiaScript provides APIs that you can use to log messages into the Service Request log.

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.

**Example:**

```javascript
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");
```
3 Custom Tasks

Cisco UCS Director:

- Provides ability to add a new workflow task with well-defined inputs and outputs.
- Uses script to implement the logic.
- Supports CloupiaScript.
- Enables export and import of custom tasks among systems.
- Enables to manage the custom tasks through the following path:
  Policies → Orchestration → Custom Tasks

Once a custom task is created, there is no inherent difference in the behavior of the built-in task and custom task. The custom task can internally invoke any other built-in task or other custom tasks. The custom task acts as a wrapper for other task or tasks.

3.1 Adding a New Custom Task

To define a new custom task, do the following:

1. Provide the basic details for the task: Task name, label, summary, and description.
2. Provide the category under which the task must appear.
3. Enable the Activate flag if you want to register the task with orchestrator and to make it immediately usable in workflow.
4. Define inputs for the task. For more details, see Custom Task Inputs.
5. Define outputs for the task. For more details, see Custom Task Outputs.
6. Create the script. For more details, see Creating the Custom Task Script.

3.1.1 Defining Custom Task Inputs

- **Input Field Name**: An internal name of the variable. It must start with an alphabetic character and must not contain spaces or special characters.
- **Input Field Label**: A label to identify the input field.
- **Input Field Type**: Choose the type of input category. The category defines how the user will be prompted for the value.
- **Map to Input Type**: (Optional) If this task input must be mappable to workflow user input, specify the type of variable that can be mapped to this variable.
- **Input Field Help**: (Optional) A description that is shown on when you hover the mouse over the field.
- **Input Field Annotation**: (Optional) Hint text for the input field.
Cisco UCS Director Cloupia Script Cookbook

- **Field Group Name**: If specified, all the fields with matching group would be put in to the field group.
- **Mandatory**: If checked, the user must provide a value for this field.

According to the input field type, the field attributes need to be defined:

- The Text Field Attributes are applicable when the input field type is **Text**.
- The LOV Field Attributes are applicable when the input field type is **LOV (List of Values)** or **LOV with Radio buttons**.
- The Table Name is applicable when the input field type is **Table or Popup Table or Table with selection checkbox**.

3.1.2 Defining Custom Task Outputs

- **Output Field Name**: A unique name of the output field. It must start with an alphabetic character and must not contain spaces or special characters.
- **Output Field Description**: Description of the output field.
- **Output Field Type**: Type of the output. This determines how the output can be mapped to other task inputs.

Internally all outputs are treated as String objects irrespective of the type of the output.

3.1.3 Creating the Custom Task Script

CloupiaScript is used to define the handler for the custom task. There are six pre-defined top level variables in any CloupiaScript: ctxt, logger, util, input, output, and metadata.

- **Object ctxt**: This variable points to the Cloupia Workflow Execution context. Using this object, the script can access information about the current workflow, task, input, and output, and can access the Cloupia APIs. Using the Cloupia APIs, you can invoke the create, read, update, and delete (CRUD) operations, workflow tasks, and expose other REST APIs.
- **Object logger**: This variable or object points to the Cloupia workflow logger object. All scripts must add reasonable amount of logging statements, which will be logged into the Service Request log.
- **Object util**: Few built-in utility methods can be accessed using this variable.
- **Object input**: This variable points to a simple POJO (object) that holds all the task input variables.
- **Object output**: This variable points to a simple POJO (object) that holds all task output variables.
Object metadata: This variable points to the task definition object, and is not used in implementation scripts.

Ensure that you do not use ctxt.setOutputValue() and ctxt.updateInput() in the custom tasks.

3.2 Exporting a Custom Task
You can export a custom workflow task to your local system to update or make any changes to the task. Then, you can import the updated custom workflow task to Cisco UCS Director.

To export a custom task, do the following:
1. Navigate to Policies > Orchestration, and click the Custom Workflow Tasks tab.
2. Choose the custom task to export.
3. Click Export Tasks.
   The Export Tasks dialog box appears.

In the Export Tasks dialog box, complete the following fields:

- Exported By: Cisco UCS Director user ID of the exporting user.
Cisco UCS Director Cloupia Script Cookbook

- **Comments**: Notes about the task or group of task that are being exported.
- **Exported File Name**: The system will auto generate the file name. Alternatively, a file name can be specified for the exported custom workflow task file.

4. Choose one or more tasks from the list of tasks to export.
5. Click **Submit**.

The Export Task action creates a file with an extension .wfdx, which can be imported to any system.

### 3.3 Importing a Custom Task

You can import a custom workflow task from any system to Cisco UCS Director after you update the custom workflow task.

To import a custom task, do the following:

1. Navigate to **Policies > Orchestration**, and click the **Custom Workflow Tasks** tab.
2. Choose the custom task to import.
3. Click **Import Tasks**.
   - The Import Tasks dialog box appears.

4. Click **Browse** to choose an exported file for upload, and click **Upload**.
5. Click **Save Tasks**. This displays the task exported time, task exported username, and comments provided when the task is exported. The imported task details appear in a table format.
6. To overwrite the existing task with the imported task, check the **Overwrite if the task exists with same name** check box.

   If the **Overwrite if the task exists with same name** check box is unchecked, a new task will be added with a number \( n \) to make the name unique.

7. Click **Import**.

**Custom Task Example: Get Submitter Details**

```javascript
function getSubmitterDetails(){
    // Get the current workflow submitter’s profile
    var userProfile = ctxt.getAPI().userAPIGetMyLoginProfile();
    output.SUBMITTER_USERID = ctxt.getUserId();
    output.SUBMITTER_FIRSTNAME = userProfile.getFirstName();
    output.SUBMITTER_LASTNAME = userProfile.getLastName();
    output.SUBMITTER_GROUP_NAME = userProfile.getGroupName();
    // output fields are always Strings, to convert ID to string
    output.SUBMITTER_GROUP_ID = ""+userProfile.getGroupId();
    output.SUBMITTER_ROLE = userProfile.getRole();
    output.SUBMITTER_EMAIL = userProfile.getEmail();
}
getSubmitterDetails();
```
Cisco UCS Director Cloupia Script Cookbook

**Details:**

- Task contains 0 inputs
- Task contains 7 outputs

SUBMITTER_USERID (Description: 'User ID of the submitter', Type: cuic_user_ID)
SUBMITTER_EMAIL (Description: 'Email address of the user', Type: email_address_list)
SUBMITTER_FIRSTNAME (Description: 'Firstname of the submitter', Type: gen_text_input)
SUBMITTER_LASTNAME (Description: 'Last name of the submitter', Type: gen_text_input)
SUBMITTER_GROUP_ID (Description: 'Group ID of the submitter', Type: userGroup)
SUBMITTER_GROUP_NAME (Description: 'Group name of the submitter', Type: gen_text_input)
SUBMITTER_ROLE (Description: 'Role of the user', Type: gen_text_input)

**Inner Tasks**

- Inner tasks produce the log messages to the current service request (SR).
- Inner tasks register object changes to the current SR.
- Inner tasks register the undo tasks to the current SR (so Rollback will work correctly).
- Inner task output is not automatically visible outside the custom task. Script developer must explicitly save it using the ‘output.X = Y’ expression.
- Each custom task can invoke multiple inner tasks.
- Each custom task may invoke an inner task in a loop.
- If inner task fails, the outer custom task automatically fails.
- Inner task can be modified by using ‘try{}catch(e){ }’ block.

    api.mapLun(input.lunpath, iniGroupName, -1, true);

**4 Handling Numbers in CloupiaScript**

You can handle the number inputs in CloupiaScript in one of the following ways:

1. Converting string variable to integer.
2. Converting int variable to integer.
4.1 Defining the Input type in the Custom Task

1. Define the Custom task input as Text

![Image showing the input type as Text]

2. Define the custom task input as Number

![Image showing the input type as Number]

CloupiaScript snippet for defining input as text and number are as follows:

1. Defined input as text

   ```javascript
   function handleNumbers(){
       var strInput=input.stringInput;
       var convertedCustomTaskStringInput = null;
       if(strInput !== null){
           // for primitives, use the java wrapper objects and pass
           // that and it will work
           convertedCustomTaskStringInput = new
           java.lang.Integer( strInput);
           logger.addInfo("convertedCustomTaskStringInput 
"="+convertedCustomTaskStringInput);
       }
   }
   handleNumbers();
   ```

This example expect the user input for the custom task. If the input is not null, the input string is converted in to number and printed in the log.
If the given input is not number, the NumberFormatException is thrown in the log.

### 2. Defined input as Number

```java
importPackage(java.lang);

function handleNumbers() {
    var strInput = input.Input * 2;
    var convertedCustomTaskStringInput = null;
    if (strInput != null) {
        // for primitives, use the java wrapper objects and pass that and it will work
        convertedCustomTaskStringInput = strInput;//new java.lang.Integer(strInput);
        logger.addInfo("convertedCustomTaskStringInput = "+convertedCustomTaskStringInput);
    }
}
handleNumbers();
```

### 5 Accessing a Network Device

A network device can be accessed using SSH.

To access a network device, do the following:

1. Create a SSHClient (com.cloupia.lib.util.ssh.SSHClient) instance by passing the device details.
2. Connect and open a session.
3. Communicate with the device using the PrintStream object.
4. Disconnect the client.

The following snippet shows how to issue a command on the device.

**Inputs of the task:**

- ipAddress = 172.25.168.221
- username = <username of the device>
- password = <password of the device>
- command = “show ip interfaces”

**Sample Script:**

```java
importPackage(com.cloupia.lib.util.ssh);
importPackage(java.io);
function testSSHClient() {
    var client = new SSHClient(input.ipAddress, 22,
        input.userName, input.password);
```
Cisco UCS Director Cloupia Script Cookbook

```javascript
client.connect();
var session = client.openShell(511,25);
var shellStream = new
PrintStream(session.getOutputStream());  // printStream for convenience
shellStream.println(input.command);
shellStream.flush();
client.disconnect();
}
testSSHClient();
```

6 Accessing a Network Element Account

The following snippet shows how to access an account of the network element.

**Sample Script:**

```java
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);
```
7 Accessing Session or API for Virtual and Physical accounts

The functionalities are defined in the Cisco UCS Director features. If required, they can be reused by accessing the feature API instead of redefining the same functionality. Also, session objects are defined in Cisco UCS Director. Create the required session object and access the API.

**NetApp**

The following snippet shows the Netapp LUN mapping functionality.

```javascript
importPackage(com.cloupia.lib.cIaaS.netapp);
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.lib.cIaaS.netapp.model);
function getNetAppApi()
{
    var filerIdentity = new NetAppFilerIdentity(input.accountName);
    var accountName = null;
    if (filerIdentity != null)
    {
        accountName = filerIdentity.getAccountName();
    }
    var account = InfrastructureDataUtil.getAccount(accountName);
    var session = new NetAppSession(account);
    var api = new NetAppAPI(session);
    logger.addInfo("API: "+api);
}
getNetAppApi();
```

**VMware**

```javascript
importPackage(com.cloupia.lib.cIaaS.vmware);
importPackage(com.cloupia.service.cIM.inframgr);

function accessVMWareAccount()
{
    var accountName = input.accountName;
    var account = InfraPersistenceUtil.getAccount(accountName);
    var si = new VCenterConnectionManager(account).getServiceInstance();
    var driver = new VCenterDriver();
}
accessVMWareAccount();
```

Use the vcenterDriver API to invoke the VMware APIs and CRUD operations.

**Example**:

```javascript
driver.updateSingleVM(Account creds, GenericVM gvm, VMWareVMSummary vmsummary, boolean includeSnapshotInfo);
```
Cisco UCS Director Cloupia Script Cookbook

driver.vmAction(Account account, VMWareManagerData data, String vmName, String action, ActionParam[] param, int requestId, ServerProfile profile, GenericVM gvm);

Using REST API:
APIProvider.getInstance().performAction(ReportContext context, VMActionUtil.ACTION_DELETE_VM_DISKS, String comment, String userId, ActionParam[] actionParams)

Invoke inventory:
var controller = com.cloupia.service.cIM.inframgr.InfraMgrImpl.getController(account.getAccountName());
controller.requestImmediatePoll();

HyperV

importPackage(com.cloupia.feature.hypervController.userApi);
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.lib.cIaaS.hyperv.model);
importPackage(com.cloupia.lib.cIaaS.hyperv.psapi);

function accessHyperVAccount()
{
    var accountName = "hyperv";
    var creds = InfraPersistenceUtil.getAccount(accountName);
    var agent = InfraPersistenceUtil.getWinRemoteAgent(creds.getPSAgentIP());
    var remoteAgent = new RemoteAgent(agent.getAddress(), agent.getPortNumber(), agent.getAccessKey());
    var targetServer = new TargetServer(creds.getHServer(), creds.getDomain() + "\" + creds.getHUserId(), creds.getHPasswd());
    var api = new SCVMMAPI(remoteAgent, targetServer,SCVMMAPI.TIME_5_MINS);
    logger.addInfo("Hyperv api: "+api);
}
accessHyperVAccount();

Example API:
api.openRemoteSession();
response=api.getResponse("get-vmmserver -computername " + creds.getHServer());
response=api.getExample("New-VirtualDiskDrive -VM \"" + vmName + "\" -SCSI -Size " + size + " -Bus " + bus + " -LUN " + lun + " -FileName " + diskName + " -" + type);
if (!HypervUtil.isScvmm2008(accountName)) {
    isStatusOk("New-SCVirtualDiskDrive", api);
} else{
```javascript
isStatusOk("New-VirtualDiskDrive", api);
}
api.closeRemoteSession();

**UCS**

importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.feature.ucsController);
importPackage(com.cloupia.lib.cIaaS.hyperv.psapi);
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.lib.cIaaS.ucs);

function accessUCSAccount(){
    var account = InfrastructureDataUtil.getAccountByType(input.accountName,InfraAccountTypes.UCSM);
    api = UcsDataPersistenceUtil.getNewUcsAPISession(account);
    logger.addInfo("Got the UCSM api: "+api);
    UcsSessionPoolManager.getInstance().closeSession(api);
}
accessUCSAccount();

**EMC**

importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.lib.cIaaS.emc.vmax.model);
importPackage(com.cloupia.lib.cIaaS.emc.vmax.);
importPackage(com.cloupia.model.cIM);
function accessEMCAccount(){
    var identity = new EMCVMAXDeviceIdentity(deviceId);
    var account = InfrastructureDataUtil.getAccountByType(identity.getAccountName(),InfraAccountTypes.EMC_VMAX);
    var vmaxapi = new com.cloupia.lib.cIaaS.emc.vmax.EmcVmaxApi();

    var account = com.cloupia.service.cIM.inframgr.InfrasctructureDataUtil.getAccountByType(accountName,
    com.cloupia.model.cIM.InfraAccountTypes.EMC_VNX);
    var api = new com.cloupia.lib.cIaaS.emc.vnx.VnxBlockXmlAPI(account);
    api.sendEmcCimRequest(String methodHdrToUse, String requestStr);
}

**Example:**
api.sendCimRequest("CreateRAIDGroupWithPowerSavingSetting")
```
Use the VNX API for the EMC VNX tasks/operations.

**Inventory:**

```java
InventoryManager.collectSubSystemInventory(account, api);
List<VNXClariionSubsystem> subsystems =
  EmcPersistenceUtil.getEMCVNXClariionSubsystemsByAccount(account.getDcName(), account.getAccountName());

InventoryManager.getBlockFullInventory(account);
```

**Whiptail**

```java
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.lib.cIaaS.whiptail.api);
importPackage(com.cloupia.lib.cIaaS.emc.vmax.);
importPackage(com.cloupia.model.cIM);

function accessEMCAccount()
{
  var account = InfrastructureDataUtil.get Account(accountName);
  var api = WarrenHillsAPI.getInstanceFor(ssrIp, account.getUserID(), account.getPassword());
  api.connect();
  api.createLUN(name, volumeGroup, size, config.getDeviceGroup());
  api.mapLUN(igName, lunName, id)
  api.performInventoryCollection(config.getAccountId(), "lun ", WTLUNs.class);
}
```

**Cisco UCS Manager Account Connection**

To configure or manage Cisco UCS, it is required to get the Cisco UCS Manager account connection.

The following sample code is for getting the Cisco UCS Manager account connection.

```java
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.feature.ucsdcontroller);
importPackage(com.cloupia.model.cIM);

function accessUCSMAccount()
{
  var account = InfrastructureDataUtil.getAccountByType(< Account Name>, infraAccountTypes.UCSM);
  var api = UcsDataPersistenceUtil.getNewUcsAPISession(account);
  var cookie = api.getLoginResponse().getOutCookie();
  var sessionId = api.getSessionId();
}

accessUCSMAccount();
```
Explanation of Fields:
1. <Account Name> – Name of the Cisco UCS Manager account.
2. InfraAccountTypes.UCSM – Type of the account.

8 Accessing reports
You can access the existing Cisco UCS Director reports and use the report data to make dynamic decision using CloupiaScript. The following script explains how to access a report and get the report data.

```javascript
importPackage(java.net);
importPackage(java.lang);
importPackage(com.vmware.vim25);
importPackage(com.vmware.vim25.mo);
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 below to create report context.
    // createContext(contextName, cloud, instanceName);
    var reportName="per.cloud.vms.paginated.config.report";
    var repContext = new ReportContext( 1, "Cloud82", "VMS-T0" );
    logger.addError("reportContext "+repContext);
    var report = getReportView(repContext, reportName);
    // You can filter report by columns
    // returns vms those Power State column value is ON
    report = report.filterRowsByColumn("Power State", "ON", false);
    logger.addInfo("report "+report.getReport);
}

accessReports();

9 Emailing Reports

You can email a Cisco UCS Director report to a user. A report can be emailed on a periodic basis by a workflow scheduler for the email workflow at the desired frequency.

The following example emails a list of all powered on VMs to a user specified in the workflow input variable: Email Address.

function sendEmail(){
    // Assume the To Email Addres is in the input variable 'Email Address'
    // var toEmail = [ ctxt.getInput("Email Address") ];
    var toEmail="sougoyal@cisco.com";
    var message = new EmailMessageRequest();
    message.setToAddr(toEmail);
    message.setSubject("VM report");
    message.setFromAddress("no-reply@cisco.com");
    var buffer = new StringWriter();
    var printer = new PrintWriter(buffer);
    var formatter = new Formatter(new File(."), printer);
    formatter.printTable(report);
    printer.close();

    sendEmail();
10 Accessing a Delegate API

You can access the delegate API by passing the namespace of delegate in the accessDelegateAPI method of APIProvider.java. This method will return the delegate object.

Example code:
In the sample code, the namespace of delegate (UserAPIChargeBack) is highlighted.

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

After getting the delegate, you can execute any method in the delegate class.

11 Accessing and Operating on a Database Table

You can access and operate a table of a Cisco UCS Director database by passing the corresponding PoJo which represents the table in the database to the getStore() method as shown in the following example code.

Example code:
In the sample code, PoJo class related to the UCS server table in the database is highlighted.

```javascript
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());
```

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

12 Provisioning a Catalog Multiple times

You need the following input to provision catalogs:

- CATALOG_ID – ID for catalogs to be provision.
- VDC_ID – ID of VDC for particular catalogs.
- PROVISION_QTY – Number of times you need to provision as integer values.
```javascript
var catId = ctxt.getInput(CATALOG_ID);
var vdcId = ctxt.getInput(VDC_ID);
var quantity = Integer.valueOf(ctxt.getInput(PROVISION_QTY));

From that input, you need to load object as follows:

```javascript
var vdc = VDCUtil.getVDC(vdcId);
var vdcName = vdc.getVdcName();
var cat = VDCUtil.getVDCCatalogItem(catId);
var catName = cat.getCatalogItemName();
``` 

Create a loop for provisioning the catalogs n number of times and use thread sleep method to delay between provisioning.

**Create an additional parameter for execution:**

Array of service request ID to see status.

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

Comment is passed as an input parameter to call method.

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

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

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

Set the begin time.

```javascript
var beginTime = -1;
for (var ctr = 0; ctr < quantity; ctr = ctr + 1) {
  logger.addInfo("Provision VM =" + (ctr+1) + " of "+qty);
  var srId = ctxt.getAPI().userAPISubmitServiceRequest(catName, vdcName, duration, beginTime, 1, comment);
  childSrIdArray[ctr] = srId;
  var milliseconds = delaySecondsBetweenInvocation * 1000;
  Thread.sleep(milliseconds);
}
```

Check the successful execution of API using the following code.

```javascript
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");
  }
}
```
Getting Inputs or Outputs of a Service Request

To view the input or output of a service request, do the following:

1. Navigate to **Organization > Service Request** and click the **Service Requests** tab.

2. Choose a service request and click **View Details**. A pop-up window appears.

3. Click **Input/Output**.

The input and output of workflow and sub workflow are displayed.
14 Calling or Executing a Workflow Task
You can invoke a workflow task from CloupiaScript using an inner task.

The following code snippet shows how to invoke a workflow task using the `ctxt` implicit object of CloupiaScript.

```javascript
// 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=thinProvision
    task.setInput("Thin Provisioning", input.thinProvision);

    // Now execute the task. If the task fails, then it will throw an exception
```
task.execute();

} // Invoke the task
Create_VM_Disk();

15 Setting a Task to Succes or Failed State

The following snippet shows how to set a task as Success or Failed.

If(true){
    ctxt.setFailed("Error output message"); //to make task
    failed upon some conditions
    ctxt.exit(); // to exit task flow normally
} else{
    ctxt.setSuccessful(); // to make task success upon some
    conditions
}

16 Invoking Another Workflow

Using the userAPISubmitWorkflowServiceRequest("workflow_name", params,
parent_srId) API, you can invoke another workflow.

- 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.

The following example explains how to invoke Task_Y from Task_X.

var params = util.createNameValueList();
params.addValuePairutil.createNameValue("Input_A", "Value_A");
params.addValuePairutil.createNameValue("Input_B", "Value_B");
params.addValuePairutil.createNameValue("Input_C", "Value_C");

var childSrId=  
ctxt.getAPI().userAPISubmitWorkflowServiceRequest("Task_Y",  
params, ctxt.getSrId());

// wait for completion of the workflow. Or a maximum of 60
seconds
// status code can be one 0 (Success), 1 (failed), -1 (Invalid SR
ID)
var status = ctxt.waitForCompletion(childSrId, 60000);
// Now that workflow is complete, we can access the output variable D of the workflow Task_Y
var output_y = ctxt.getOutput("Task_Y.D", childSrId);

The Input_Name provided in the parameter list must be same as the Input_Name defined in the workflow to be invoked.

### 17 Accessing a Feature Class

You have to load the class using the FeatureContainer.

Example:

```javascript
var container = FeatureContainer.getInstance();
var cl = container.loadClass("com.cloupia.feature.cloudsense.CloudSenseUtil");
var csUtil = cl.newInstance();
var mailSettings = csUtil.getMailSettings();
logger.addInfo("Got mail settings: "+mailSettings);
```

### 18 Calling an API on a Remote Cisco UCS Director

You can call an API on a remote Cisco UCS Director by selecting the required server IP address in the API browser.

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

Example:

To get a tabular report:

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

To get all catalogs:

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

To get all VM actions:

```javascript
client.sendJSONRequest("userAPIGetAvailableVMActions", Arrays.asList(new String[] {"2293"}));
```
To submit a service request:

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

To execute a workflow:

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

19 Critical Section

Critical section is like a synchronized block where only one thread is allowed to enter and execute the critical section at a time using a locking mechanism.

The critical section API is:

```java
public static void enterCriticalSection(CustomActionTriggerContext context,
CustomActionLogger logger, String lockName) throws Exception
```

When the workflow thread running with a context srId wants exclusive access to specific block or section with a specified sectionName or lockName, the critical section API is used for gaining exclusive access to the critical section. If any other thread wants to enter the critical section, it has to wait till the thread holding the lock releases it with the exitCriticalSection API.

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

```java
public static void exitCriticalSection(CustomActionTriggerContext context,
CustomActionLogger logger) throws Exception
```

20 Getting Complete List of VLANs across PODs

<table>
<thead>
<tr>
<th>Input</th>
<th>POD Or Data center Or Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>POD1 --&gt; v1,v2, v10</td>
</tr>
<tr>
<td></td>
<td>POD2 -&gt; v11, v13 etc.</td>
</tr>
<tr>
<td></td>
<td>Total vLANS Used = 20.</td>
</tr>
<tr>
<td></td>
<td>Pools (vP1, vP2).</td>
</tr>
</tbody>
</table>

Sample Script:

```javascript
The UCS cloud is implemented.

var accountName = input.UCSCLOUD;
```
21 Getting Complete List of IPs used per-pod or across

Input : POD name.
Output : Comma separated list of all the IPs on that pod.

Sample Script :

```java
importPackage(java.lang);
importPackage(java.util);
importPackage(com.cloupia.model.cIM);

function getListOfIPs(){
    var allAcct = ctxt.getAPI().getAllAccounts();

    var listOfIP = new ArrayList();
    for (var i = 0; i < allAcct.length; i++) {
        var acc = allAcct[i];
        if(acc.getAccountType()==Account.AMAZON_EC2){
            listOfIP.add(acc.getVServer());
        }else if(acc.getAccountType()==Account.VMWARE){
            listOfIP.add(acc.getHServer());
        }else if(acc.getAccountType()==Account.HYPERV){
            listOfIP.add(acc.getHServer());
        }else if(acc.getAccountType()==Account.XENDESKTOP){
            listOfIP.add(acc.getXenCtrlIP());
        }
    }
    output.IPList = listOfIP;
}
getListOfIPs();
```

Sample Script

To lock and unlock a vDC
Input : VDC Name
Output : None

```java
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);
    flag=VDCUtil.modifyVDC(vdc);
  }
  catch (e)
  { 
    logger.addError("Got Exception When Trying To Modify vDC.");
    ctxt.setFailed(e);
    ctxt.exit();
  }
  if(flag) 
  { 
    logger.addInfo("Successfully Modify vDC. ");
    vdc=VDCUtil.getVDC(vdcId);
    output.vdcId=vdcId;
    output.isLocked=vdc.isLocked();
  }
  else
    logger.addInfo("Unable to Modify vDC .");
}
lock();

22 Creating a Task to get the List of Hosts from a VMware cluster

   Input : VMware account, VMware cluster name.
   Output : Comma separated hosts that can be mapped to the Associate VNX LUN as Datastore task. However, this operation is applicable on all hosts in a cluster, not just a single host.

Sample Script :

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

    function getListOfHosts(){
      var listofHosts = 
      InfraPersistenceUtil.getVMWareHostsByCluster(input.VMWAREACCOUNT, input.CLUSTER);
      return listofHosts;
    }
23 Moving Multi-VM across vDC's

Input: VDC-one, VDC-two.
Output: All VMs which were in VDC-one are now in VDC-two.

Sample Script:

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

function moveVMsAcrossVDC(){
var sourceVdc = input.sourceVDC;
var sourceVdcId = parseInt(sourceVdc);
var sourceVdcName = VDCUtil.getVDC(sourceVdcId).getVdcName();
logger.addInfo("vDC:"+sourceVdcName + "--> vDC Id:
"+sourceVdcId);
var destinationVdc = input.destinationVDC;
var destinationVdcId = parseInt(destinationVdc);
var destinationVdcName = VDCUtil.getVDC(destinationVdcId).getVdcName();
logger.addInfo("vDC:"+destinationVdcName+"--> vDC Id:
"+destinationVdcId);
var vmLists = VDCUtil.getVMsAssociatedWithVDC(sourceVdcId);

for(var i=0; i<vmLists.size();i++){
    var vmId = vmLists.get(i);
    logger.addInfo("vmId:"+vmId);
    try{
        var vmData = GroupManagerImpl.getVMDatasByVМИd(vmId);
        var vmsummary = ctxt.getAPI().getVMBasicInfo(vmId);
        var vmType = vmsummary.getVmType();
        var vmName = "";
        if(vmType.equals("VMWare"))
            vmName = ctxt.getAPI().getVMwareVMInfo(vmId).getName();
        else if(vmType.equals("Hyper-V"))
            vmName = InfraPersistenceUtil.getHyperVVMSummary(vmId).getName();
        if (vmData != null)
            {logger.addInfo("Assigning VM:"+vmName+" to vDC:"+vdcName);
            vmData.setVdcId(destinationVdcId);
            GroupManagerImpl.assignGroupToVM(vmData);
        }else{
            logger.warn("The VM has no assignment (VM User Data ")
            vmData = new VMUserData();
            vmData.setVmId(vmId);
            vmData.setVdcId(destinationVdcId);
            GroupManagerImpl.assignGroupToVM(vmData);
        }
    }catch(e){
```
```javascript
```
logger.addError("Error occurred while assigning VMs to vDC");
ctxt.setFailed("Task failed to assign VM to vDC");
ctxt.exit();
}
}
output.Result = " VMs successfully assigned to destination vDC:" + destinationVdc;
}

24 Rolling back a Task in a Workflow

Rollback / Undo of a task in a workflow

If you want to rollback a task in a workflow, use the change tracker API, and take care of rollback of the task.

Example Script:

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();

The undo task handler name and undo config object are the important parameters for executing 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 undo of modify/delete operations:

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