Examples

This chapter contains the following sections:

Logging in CloupiaScript

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 Cisoc UCS Manager.

Code

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

For NetApp accounts:

```
function netappAccounts(){
var opName = "netapp:userAPIGetNetAppAccounts";
var payload = {};
var jsonInString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName,jsonInString);
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 jsonInString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName,jsonInString);
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 jsonInString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName,jsonInString);
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 jsonInString = JSON2.stringify(payload);
var result = ctxt.getAPI().performOperationOnJSONPayload(opName,jsonInString);
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 coloumns.
//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 delegateOdj = api.getAPIDelegate("chargeback:userAPICheckFunds");
}
accessDelegateAPI();
```

Results

The above sample returns the delegate API implementation instance (delegateOdj). 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 reqeust 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 srId) 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.javaToJsonString(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.addRequestHeader("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 acess 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.setToAddrs(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 an 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.