



## 例

---

この章は、次の項で構成されています。

- [CloupiaScript へのログイン, 2 ページ](#)
- [数値入力処理, 3 ページ](#)
- [ネットワーク デバイスへの SSH, 4 ページ](#)
- [ネットワーク要素アカウントへのアクセス, 5 ページ](#)
- [仮想アカウントと物理アカウントの表示, 6 ページ](#)
- [レポートへのアクセス, 9 ページ](#)
- [デリゲート API へのアクセス, 11 ページ](#)
- [データベース テーブルへのアクセスと操作, 11 ページ](#)
- [複数回の仮想マシンのプロビジョニング, 12 ページ](#)
- [ワークフロー タスクの呼び出しまたは実行, 16 ページ](#)
- [タスクの \[成功 \(Successful\)\] または \[失敗 \(Failed\)\] ステータスへの設定, 17 ページ](#)
- [その他のワークフローの呼び出し, 18 ページ](#)
- [リモート Cisco UCS Director 上での API の呼び出し, 19 ページ](#)
- [クリティカルセクションの実行, 21 ページ](#)
- [ポッド全体での VLAN の完全なリストの取得, 22 ページ](#)
- [ポッドごとに使用される IP の完全なリストの取得, 23 ページ](#)
- [VDC のロックまたはロック解除, 24 ページ](#)
- [VMware クラスタからホストのリストを取得するタスクの作成, 25 ページ](#)
- [VDC 間の複数の VM の移動, 25 ページ](#)
- [ワークフロー内のタスクのロールバック, 28 ページ](#)
- [ServiceNow チケット生成との統合, 29 ページ](#)

- [Cloupia スクリプトからメールを送信, 31 ページ](#)
- [古いサービス リクエストのアーカイブの自動化, 32 ページ](#)
- [ワークフロー送信者情報の決定, 33 ページ](#)
- [仮想マシン ディスクのサイズ変更, 35 ページ](#)
- [Jar ファイルのアップロード, 36 ページ](#)
- [カスタム タスクのライブラリの呼び出し, 37 ページ](#)
- [カスタム タスクでの登録済み値のリスト \(LOV\) の使用, 38 ページ](#)
- [カスタム ワークフロー タスクでの表形式のレポートの使用, 39 ページ](#)

## CloupiaScript へのログイン

### 目標

サービス リクエストのログにメッセージを出力します。

### コンテキスト

出力ログ ステートメントを表示する場合。

### 前提条件

なし

### コンポーネント

Logger オブジェクトは、次のロギング用のメソッドを提供します。

- `addDebug` : デバッグ メッセージを灰色で表示します。
- `addInfo` : 通常のメッセージを黒色で表示します。
- `addWarning` : 警告メッセージをオレンジ色で表示します。
- `addError` : エラー メッセージを赤色で表示します。

### コード

```
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.");
```

### 結果

Logger オブジェクトに渡されるメッセージが出力されます。

### 実装

このコードを実行するために必要な変更はありません。

## 数値入力の処理

### 目標

カスタム タスクの数値入力を処理します。

### 前提条件

カスタム タスクは、テキストまたは数値入力で Cisco UCS Director で最初に定義する必要があります。

### コンポーネント

カスタム タスク ライブラリは必要ではありません。

### コード

```
// 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();
```

### 結果

このスクリプトでは、ログに入力値が出力されます。

## 実装

実装は簡単です。カスタム タスクの組み込み `input` オブジェクトから入力変数を取得して、`Java Integer` ラッパー オブジェクトに引数として指定します。指定された入力が数字でない場合は、`NumberFormatException` エラーがスローされログに表示されます。

# ネットワーク デバイスへの SSH

## 目標

SSH を使用してネットワーク デバイスにアクセスします。

## コンテキスト

SSH を使用してデバイス (VM など) にログインし、デバイスでコマンドを実行するためのカスタム タスクが必要な場合。

## 前提条件

SSH 対応デバイスが、既知の IP アドレス、ユーザ名、およびパスワードを使用してネットワークで利用可能である必要があります。カスタム タスクは次の入力で定義されます。

- `ipAddress` (リモート デバイスの IP アドレス)
- `username` (デバイスでの)
- `password` (デバイスでの)
- `command` (デバイスに発行するためのコマンド)

## コンポーネント

`com.cloupia.lib.util.ssh.SSHClient` : この API はリモート SSH サーバまたはデバイスに接続してコマンドを実行し、結果を標準出力 (stdout) に示します。

## コード

```
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();
```

### 結果

SSH 接続が開き、コマンドがリモート デバイスに送信されます。その後、接続が閉じられます。

### 実装

コード例の `client.openShell()` メソッドは、サードパーティのライブラリ API `com.maverick.ssh.SshSession.requestPseudoTerminal()` を内部的に使用します。

このスクリプトは、コマンドの結果を確認しません。実際、セッションの入力ストリームは全く開かれず、コマンドの送信後は単に切断されます。実稼働環境でこのスクリプトを実行した場合は、より詳しいエラー チェックを行いたいのは当然です。

## ネットワーク要素アカウントへのアクセス

### 目標

ネットワーク デバイスを接続し、CLI コマンドを送信します。

### コンテキスト

CLI コマンドをデバイスに送信する場合。

### 前提条件

ネットワーク デバイスが UCS Director に追加されている必要があります。

### コンポーネント

- `NetworkDeviceManager` : このコンポーネントは、引数としてデバイスの詳細を受け入れ、デバイスへの接続を確立します。接続が作成された後、ユーザはコマンドを送信することでネットワーク デバイスを管理できます。
- `com.cloupia.lib.cIaaS.network.model.DeviceCredential` : 「DeviceCredential」 API はデバイスの詳細を保持します。

## コード

```
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);
```

## 結果

コマンドがデバイス上で実行され、コマンド出力がログに示されます。

## 実装

このコードを実行するために必要な変更はありません。

# 仮想アカウントと物理アカウントの表示

## 目標

仮想アカウントと物理アカウントの詳細を表示します。

## コンテキスト

アカウントに対してアクションを実行する前に、仮想アカウントと物理アカウントの名前および構成を表示する。

## 前提条件

仮想アカウントまたは物理アカウントを Cisco UCS Director に追加する必要があります。

## コンポーネント

netapp:userAPIGetNetAppAccounts API : この API は、NetApp デバイスの詳細を取得します。

ucsm:userAPIGetUCSMAccounts : この API は、Cisco UCS Manager の詳細を取得します。

コード

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

**NetApp アカウントの場合：**

```
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();
```

**Cisco UCS Manager アカウントの場合：**

```
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();
```

アカウント名を渡すことでアカウントタイプを取得します。

```
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();
```

**userAPIExecuteVMAction API** を使用して VMware VM の操作を実行します。

```
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();
```

**HyperV SCVMM の ID** を読み取ります。

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

すべての物理アカウントを読み取ります。

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



```
}
readPhysicalAccounts();
すべての仮想アカウント（クラウド）を読み取ります。

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

### 結果

仮想アカウントと物理アカウントの詳細が表示されます。

### 実装

アカウントの詳細を読み取る機能は、Cisco UCS Director に定義されています。必要に応じて、同じ機能を再定義する代わりに、機能 API にアクセスすることによって再利用できます。

## レポートへのアクセス

### 目標

既存の Cisco UCS Director のレポートにアクセスして、レポートデータをフィルタリングします。

### コンテキスト

レポートを取得し、要件ごとにレポート内のデータをフィルタリングする場合。

### 前提条件

レポートを取得するにはレポート名とコンテキストが必要です。

### コンポーネント

`ctxt.getAPI().getConfigTableReport` : このメソッドは、レポートの詳細を含む `TabularReport` オブジェクトを返します。

`ctxt.getAPI().getTabularReport` : このメソッドは、レポートの詳細を含む `TabularReport` オブジェクトを返します。

## コード

```
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 the following code to create report context
var reportName="per.cloud.vms.paginated.config.report";
//to create a context, pass the following arguments in the given order: contextName,
cloud, instanceName
var repContext = new ReportContext( 1, "Cloud82","VMS-T0 ");
logger.addError("reportContext "+repContext);
var report = getReportView(repContext, reportName);
//you can filter the report by coloumns
//to return the VMs in ON power state, use the following line
report = report.filterRowsByColumn("Power State", "ON", false);
logger.addInfo("report "+report.getReport);
}

accessReports();
```

## 結果

このコードでは、VMのレポートが取得され、「オン」の電源状態に応じてデータがフィルタリングされます。

### 実装

このコードを実行するために必要な変更はありません。

## デリゲート API へのアクセス

### 目標

APIProvider.java の accessDelegateAPI メソッドでデリゲートの名前空間を渡すことによって、デリゲート API にアクセスします。

### コンテキスト

名前空間を使用してデリゲート API にアクセスし、操作を実行する場合。

### 前提条件

デリゲート API の実装インスタンスにアクセスするには、名前空間が必要です。

### コード

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

### 結果

上の例では、デリゲート API の実装インスタンス (delegateObj) が返されます。デリゲートクラスで任意のメソッドを実行できます。

### 実装

このコードを実行するために必要な変更はありません。

## データベース テーブルのアクセスと操作

### 目標

対応する PoJo を渡すことで、Cisco UCS Director データベースのテーブルにアクセスし、操作を行います。(PoJo はデータベース テーブルを表します。)

### コンテキスト

データベース テーブルにアクセスし、詳細を取得するためのスクリプトが必要な場合。

### 前提条件

情報を取得するには、有効な PoJo クラス名と有効なクエリが必要です。

### コンポーネント

ObjStoreHelper.getStore : このメソッドはデータベース テーブルにアクセスして詳細を取得します。

### コード

```
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());
```

### 結果

ObjStoreHelper クラスの getStore() メソッドは、モデル クラスの名前を入力として取得し、対応するオブジェクトストアを出力として返します。

### 実装

このコードを実行するために必要な変更はありません。

## 複数回の仮想マシンのプロビジョニング

### 目標

仮想マシン (VM) を複数回プロビジョニングします。

### コンテキスト

仮想データセンター (VDC) で VM を  $n$  回プロビジョニングします。

### 前提条件

VM を複数回プロビジョニングするために次の詳細を入力します。

- vDcName : VM をプロビジョニングする必要がある VDC の名前。
- catalogName : VM のプロビジョニングに使用するカタログの名前。
- qty : VM をプロビジョニングする必要がある回数。
- waitTime : VM のプロビジョニングの間に待機する時間。

### コンポーネント

- `provisionVM(vDcName, catalogName, qty, waitTime)` 関数：この関数を実行して、VDC で  $n$  回 VM をプロビジョニングします。
- `userAPISubmitServiceRequest` API：VM を作成するためのサービス リクエストを送信します。

コード

```
importPackage(java.lang);
loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
入力から、次のオブジェクトをロードする必要があります。
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
```

API レスポンスからフェッチされるサービスリクエスト ID (SRID) の配列を格納する配列を設定します。

```
var childSrIdArray = [];
ロギングメッセージのロガーメソッドを定義します。
```

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

VM のプロビジョニングの回数をゼロに設定する場合、エラーメッセージを表示するように `addError` メソッドを定義します。

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

`provisionVM()` 関数を使用して VM を複数回プロビジョニングします。

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

`userAPISubmitServiceRequest` API を使用して VM を作成します。

```
var opName = "userAPISubmitServiceRequest";
コメントはメソッドを呼び出すための入力パラメータとして渡されます。
```

```
var comment = "";
VM の持続時間を -1 (または時間数) として設定します。
```

```
var duration = -1;
開始時刻を設定します。
```

```
var beginTime = -1;
userAPISubmitServiceRequest API のペイロードを作成します。
```

```
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);
VM を n 回プロビジョニングするためのループを作成します。
```

```
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();
}
}
```

スレッドのスリープメソッドを使用してプロビジョニング間を遅延させます。

```
var srId = parseResponse.serviceResult;
childSrIdArray[ctr] = srId;
var milliseconds = waitTime * 1000;
Thread.sleep(milliseconds);
}
```

SR ID の配列のループを作成し、SR ステータス（成功または失敗）を取得します。

```
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");
}
}
}
```

try および catch ステートメントを使用して、provisionVM 関数を呼び出します。

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

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

## 結果

VM は、*qty* 変数で定義されているとおりに、VDC で *n* 回プロビジョニングされます。

## 実装

コードを実行するために必要な場合がある変更については、上記のコードを参照してください。

# ワークフロータスクの呼び出しまたは実行

## 目標

内部タスクを使用して CloupiasScript からワークフロータスクを呼び出します。

## 前提条件

内部タスク名と入力のセットが必要です。

## コンポーネント

ctxt.createInnerTaskContext(): このメソッドを使用して、タスク変数にアクセスし、入力値を設定します。



## コード

```
// 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();
```

## 結果

このスクリプトでは、内部タスクとして VM タスクの作成が実行されます。

## 実装

このコードを実行するために必要な変更はありません。

# タスクの [成功 (Successful) ] または [失敗 (Failed) ] ステータスへの設定

## 目標

特定の条件に基づいてタスクのステータスを [成功 (Successful) ] または [失敗 (Failed) ] に設定します。

## 前提条件

なし

## コンポーネント

ctxt : ワークフロー タスクのステータス メッセージを設定するために使用されます。

### コード

```
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
}
```

### 結果

特定の条件に基づいてタスクのステータスを [失敗 (Failed) ] または [成功 (Successful) ] に設定します。

### 実装

このコードを実行するために必要な変更はありません。

## その他のワークフローの呼び出し

### 目標

別のワークフローからワークフローを呼び出します。

### コンテキスト

CloupiaScript を使用して別のワークフローを呼び出す。

### 前提条件

ワークフローの名前およびワークフロー パラメータが必要です。

### コンポーネント

`userAPISubmitWorkflowServiceRequest("workflow_name", params, parent_srId)` API を使用して、別のワークフローを呼び出すことができます。次のパラメータが必須です。

- `workflow_name` : 呼び出すワークフローの名前。
- `params` : 呼び出すワークフローに渡すパラメータのリスト。
- `parent_srId` : 子ワークフローをサービス要求 ID にリンクします。

## コード

```
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();
```

## 結果

上記のスクリプトでは、所定の入力を使用して別のワークフローが呼び出され、サービス要求が生成されます。

## 実装

inputname1 と inputname2 で指定する Input\_Name は、呼び出すワークフローで定義された Input\_Name と同じにする必要があります。呼び出すワークフローを子とするには、パラメータ parent\_srId が必要です。

子のサービス要求は、サービス リクエスト ログに表示されません。別個のサービス要求としてワークフローを呼び出す場合は、parent\_srId を -1 として渡します。

# リモート Cisco UCS Director 上での API の呼び出し

## 目標

API ブラウザで必要なサーバの IP アドレスを選択して、API をリモートで実行します。

## コンテキスト

リモートの Cisco UCS Director で API を呼び出す場合。

## 前提条件

リモート UCSD サーバが必要です。

## コンポーネント

JsonUtil : RestAPI コールをリモートで送信します。

CloupiaClient : 接続を作成します。

## コード

```
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.
```

表形式のレポートを取得するには :

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

すべてのカタログを取得するには :

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

すべての VM アクションを取得するには :

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

サービス要求を送信するには :

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

ワークフローを実行するには :

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

## 結果

このスクリプトでは、リモートの UCS Director から API が呼び出されます。

## 実装

CloupiaClient メソッドにサーバの IP アドレスとポート番号を指定します。

# クリティカルセクションの実行

## 目標

CriticalSection API を使用してクリティカルセクションを実行するためのスレッドへの排他アクセスを可能にします。

## コンテキスト

CriticalSection API を使用して、ワークフロー スレッド（コンテキスト srId を使用して実行している）は指定された sectionName または lockName を使用してクリティカルセクションに排他的にアクセスできます。クリティカルセクションにアクセスしたい他のスレッドは、exitCriticalSection API がロックを保持しているスレッドからセクションを解放するまで待機する必要があります。

## 前提条件

なし

## コンポーネント

CriticalSectionUtil : クリティカルセクションを保護するためのクラスを使用します。

## コード

クリティカルセクションを入力し、ロックを効率的に取得するには、以下を使用します。

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

スレッドがサービス要求によって保持されたすべてのロックを解放するには、次の API を使用します。

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

## 結果

複数の要求がある場合、上記の API によって 1 つずつ要求を実行できます。

## 実装

このコードを実行するために必要な変更はありません。

## ポッド全体での VLAN の完全なリストの取得

### 目標

アカウントで使用可能なすべての VLAN ID を取得します。

### コンテキスト

UCS デバイスで設定されたすべての VLAN ID を取得する場合。

### 前提条件

UCSM アカウントにアクセスできる必要があります。

### コンポーネント

UcsDataPersistenceUtil : このクラスは VLAN ID を取得します。

### コード

```
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;
```

### 結果

カンマで区切られた VLAN ID。例 : DR\_UCSM30:1,1,205,1135,1136,1001

### 実装

このコードを実行するために必要な変更はありません。

# ポッドごとに使用される IP の完全なリストの取得

## 目標

1 つのポッドで使用可能なすべての仮想アカウント IP を取得します。

## コンテキスト

ポッド名が分かっている、そのポッドからすべての仮想アカウント IP を取得する場合。

## 前提条件

仮想アカウントがポッドで使用できる必要があります。

## コンポーネント

`InfraPersistenceUtil.getAllFlexPodAccounts` : ポッドに追加されたすべてのアカウント名を返します。

## コード

```
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.getServer()+", ";
    }
}
listOfIPsPerPod = listOfIPsPerPod.substring(0, listOfIPsPerPod.length()-1);
logger.addInfo(podName+": "+listOfIPsPerPod);
output.listOfIPsPerPod = listOfIPsPerPod;
getListOfIPs();
```

## 結果

ポッド内のすべての IP のカンマ区切りリストが出力されます。たとえば、172.29.110.196、172.25.168.80、172.29.109.82 です。

## 実装

このコードを実行するために必要な変更はありません。

# VDCのロックまたはロック解除

## 目標

VDCのロックまたはロック解除。

## 前提条件

VDCを作成する必要があります。

## コンポーネント

VDCUtil : VDCをロックするにはこのメソッドを使用します。

VDCUtil.setLocked() : VDCをロックまたはロック解除するにはこのメソッドを使用します。

## コード

```
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();
```

## 結果

このスクリプトでは、VDCがロックされます。これは、UCS Directorで確認できます。

## 実装

このコードを実行するために必要な変更はありません。



# VMware クラスタからホストのリストを取得するタスクの作成

## 目標

VMware クラスタからホストのリストを取得します。

## コンテキスト

VMware クラスタ内のすべてのホストを表示する場合。

## 前提条件

VMware クラスタが使用可能である必要があります。

## コンポーネント

InfraPersistenceUtil : このオブジェクトは、VMware クラスタからホストのリストを取得します。

## コード

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

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

## 結果

カンマで区切ったホストがデータストア タスクとして関連 VNX LUN にマップされます。ただし、この操作はクラスタ内のすべてのホスト（1つのホストだけではなく）に適用されます。

## 実装

このコードを実行するために必要な変更はありません。

# VDC 間の複数の VM の移動

## 目標

ある VDC から別の VDC にすべての VM を移動します。

### 前提条件

VDC で VM が移動できる必要があります。

### コンポーネント

- `createMoResource(moveVmsResourcePath, requestPayloadStr)` : VDC に VM を移動するには、このメソッドを使用します。

## コード

```

loadLibrary("JSON-JS-Module/JSON-JS-ModuleLibrary");
VDC に VM を移動するリソースパスを設定します。

var moveVmsResourcePath = "/vmwareVM";
アカウント名、VM ID、およびユーザからの VDC 名を取得します。

var accountName = input.accountName;
var vmId = input.vmId;
var vdcName = input.vdcName;
ロギングメッセージのロガーメソッドを定義します。

logger.addInfo("Cloud Name : "+input.accountName);
logger.addInfo("VM/s Id : "+input.vmId);
logger.addInfo("vDC Name : "+input.vdcName);
moveVmsTovDC 関数を使用して、VM を VDC に移動するリクエストを作成します。

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;
ペイロードのリクエストを作成します。

var requestPayloadStr = JSON2.stringify(requestPayload);
createMoResource メソッドを使用してペイロードを実行します。

var res = ctxt.getAPI().createMoResource(moveVmsResourcePath, requestPayloadStr);
logger.addInfo("Response : " + res);
var resObj = JSON2.parse(res);
errorMessage の処理状態を確認します。

var opStatus = resObj.cuicOperationResponse.operationStatus;
if( opStatus != 0){
var errorMessage = resObj.cuicOperationResponse.errorMessage;
logger.addError(errorMessage);
ctxt.setFailed(errorMessage);
ctxt.exit();
}
}
例外を処理するために、moveVmsTovDC 関数を呼び出し、try および catch ステートメントを使用します。

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

```

## 結果

VDC 内で選択した VM が別の VDC に移動されます。

## 実装

コードを実行するために必要な場合がある変更については、上記のコードを参照してください。

# ワークフロー内のタスクのロールバック

## 目標

変更トラッカー API を使用してワークフロー内のタスクをロールバックします。

## コンテキスト

操作を実行した後にそれを取り消したい場合。たとえば、VM を作成した後に、VM をロールバックして削除したい場合。

## 前提条件

ワークフローが正常に実行されている必要があります。

## コンポーネント

タスク取り消しハンドラ名と設定取り消しオブジェクトは、タスク取り消しを実行するための重要なパラメータです。タスク取り消しには、オリジナルのタスクによって実行されたアクションを取り消すためにオリジナルのタスクに関するデータが必要です。

変更/削除操作の取り消しに関連する API : `ChangeTracker.undoableResourceModified` と `ChangeTracker.undoableResourceDeleted`

どちらの API コールも同じパラメータのセットを持っています。ワークフロー全体のロールバックを成功させるためには、ワークフロー内の各タスクに変更トラッカー API を使用した取り消しサポートが必要です。

## コード

```
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();
```

## 結果

指定された ID を持つグループがタスクの実行後に削除されます。

## 実装

このコードを実行するために必要な変更はありません。

# ServiceNow チケット生成との統合

## 目標

ServiceNow を使用してデモ インスタンスに関するデータをリセットします。

## コンテキスト

ServiceNow は、IT サービスの管理をサポートし、共通のビジネス プロセスを自動化するソフトウェア プラットフォームです。この Software as a Service (SaaS) プラットフォームには、多数のモジュラ アプリケーションが含まれています。含まれているアプリケーションはインスタンスやユーザによって異なる場合があります。HttpClient API を使用して HTTP 要求を送信し、HTTP 応答を処理します。

ServiceNow には、テストに使用できるいくつかのデモ インスタンスが用意されています。デモのログイン名およびパスワードは admin/admin です。デモのログイン名およびパスワードは定期的に変更されるため、ServiceNow で確認してください。

## 前提条件

ServiceNow はそれぞれのデモ インスタンスのデータを毎日リセットします。この API を動作させるには JSON プラグインを ServiceNow で有効にする必要があります。プラグインセクションをアクティブ化する方法については、Service Now Website を参照してください。

## コンポーネント

HttpClient : ServiceNow ソフトウェアと通信するために使用されます。

## コード

```

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.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();
}
}

```

## 結果

このスクリプトでは、ServiceNow ソフトウェアと通信します。

### 実装

スクリプトを実行する前にサーバアドレスおよびユーザ名/パスワードを指定します。

## Cloupia スクリプトからメールを送信

### 目標

特定の操作の完了後に電子メールを送信します。

### コンテキスト

特定の操作の完了後に電子メールを送信する。たとえば、VMがプロビジョニングされた後に、電子メールを送信したい場合などです。

### 前提条件

レポート値が[電子メール設定 (Mail Setup)]画面で空でないことを確認します。[電子メール設定 (Mail Setup)]画面にアクセスするには、Cisco UCS Director にログインし、[管理 (Administration)]>[システム (System)]を選択して[電子メール設定 (Mail Setup)]をクリックします。

### コンポーネント

MailManager : 電子メールの送信に使用されます。

## コード

```
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);
```

## 結果

電子メールがそれぞれの電子メール受信者に送信されます。

## 実装

このコードを実行するために必要な変更はありません。

# 古いサービス リクエストのアーカイブの自動化

## 目標

30 日を経過したすべてのサービス要求 (SR) をアーカイブする Cloupia スクリプトを設定します。

## コンテキスト

SR ID の数が多くて UI で SR ID を非表示にしたい場合。直近 30 日の SR のみを日次レポートに表示するワークフローを実行したい場合。



### 前提条件

サービス リクエスト ID を作成する必要があります。

### コンポーネント

`WorkflowManager.getInstance().archive()` : このメソッドは、SR ID をアーカイブするために使用されます。

### コード

```
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);
    }
}
```

### 結果

ワークフローの実行後に SR ID がアーカイブされ、UI に表示されません。

### 実装

行 `getOlderSRs()` に引数として日数を渡す必要があります。上記の例では、30 日を経過したすべてのサービス要求がアーカイブされます。

## ワークフロー送信者情報の決定

### 目標

ワークフローを送信した人物のユーザの詳細（名、姓、電子メールアドレスなど）にアクセスします。

## コンテキスト

ワークフローを送信した人物の詳細（ユーザ ID、名、姓、電子メール ID など）を知りたい場合。

## 前提条件

なし

## コンポーネント

APILoginProfile : ctxt.getAPI(). および userAPIGetMyLoginProfile() メソッドは、ユーザの詳細が含まれている APILoginProfile オブジェクトを返します。

スクリプトで取得されて他のタスクで使用される情報のキャプチャおよび保存用に、次の例では以下のワークフロー レベル変数を使用します。

- SUBMITTER\_EMAIL
- SUBMITTER\_FIRSTNAME
- SUBMITTER\_LASTNAME
- SUBMITTER\_GROUPNAME

## コード

```
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);
```

## 結果

このスクリプトの例では、ワークフローの送信者のユーザ ID、名前、電子メール、グループ、およびロールが出力されます。

## 実装

このコードを実行するために必要な変更はありません。

# 仮想マシン ディスクのサイズ変更

## 目標

プロビジョニング後に VM のディスク サイズを変更します。

## コンテキスト

## 前提条件

VMにディスクが必要です。ライブラリ タスク「VMディスクのサイズ変更」を参照できます。

## コンポーネント

VMWareVMSummary : ctxt.getAPI().getVMwareVMInfo(vmid) メソッドはオブジェクト VMWareVMSummary を返します。

ctxt.getAPI().performAction() : VM ディスクのサイズ変更操作を実行します。

vmid : サイズ変更が必要な VM を指す入力変数。

## コード

```
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");
```

## 結果

スクリプトの実行後に VM ディスクのサイズが変更されます。

## 実装

このコードを実行するために必要な変更はありません。

## 関連項目

UCS Director 内の追加の組み込みライブラリ タスクを参照してください。

# Jar ファイルのアップロード

## 目標

スクリプト モジュールから jar ファイルをアップロードして、外部 jar ファイルをカスタム タスクで使用可能にします。

## コンテキスト

カスタム タスク用の外部 jar ファイルにアクセスする。

## 前提条件

Cisco UCS Director のスクリプト モジュールを使用して、jar ファイルが登録されていることを確認します。『*Cisco UCS Director Orchestration Guide*』の「Using Script Modules」の章を参照してください。

## コンポーネント

スクリプト モジュール

## コード

サンプルの jar ファイル「Calculatedemo.jar」に次のコードが存在すると仮定します。

```
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;
    }
}
```

次のスクリプトでは、Calculatedemo.jar ファイルを使用したカスタム タスクを完了できません。

```
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);
```

### 結果

アップロードされた jar ファイルを、カスタム タスクから正常にアクセスできます。

### 実装

このコードを実行するために必要な変更はありません。

## カスタム タスクのライブラリの呼び出し

### 目標

カスタム タスクのライブラリ関数を呼び出します。

### コンテキスト

再利用可能なコードがあり、複数のカスタム タスクで使用する。ライブラリに再利用可能なコードを追加し、カスタム タスクからライブラリを呼び出すことができます。

### 前提条件

Cisco UCS Director で再利用可能なコードでライブラリを作成します。『*Cisco UCS Director Orchestration Guide*』の「Using Script Modules」の章を参照してください。

### コンポーネント

スクリプト モジュール

### コード

ライブラリ「lib1」に次の再利用可能なコードが存在すると仮定します。

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

次のスクリプトを使用して、ライブラリを呼び出します。

```
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);
```

### 結果

ライブラリがアクセスされ、カスタム タスクが正常に実行されます。

## 実装

このコードを実行するために必要な変更はありません。

# カスタム タスクでの登録済み値のリスト (LOV) の使用

## 目標

カスタム タスクから登録済み LOV にアクセスします。

## コンテキスト

## 前提条件

Cisco UCS Director に値のリスト (LOV) を登録します。『*Cisco UCS Director Orchestration Guide*』の「Using Script Modules」の章を参照してください。

## コンポーネント

スクリプト モジュール

## コード

```
{
  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;
}
```

## 結果

このスクリプトでは、カスタム タスクから LOV のサイト リストにアクセスできます。

## 実装

カスタムタスクからLOVにアクセスするには、カスタムタスクを作成し、変数を定義します。変数名はスクリプトモジュールで作成したLOVと同じである必要があります。

# カスタムワークフロータスクでの表形式のレポートの使用

## 目標

カスタムタスクから登録済み表形式レポートにアクセスします。

## 前提条件

表形式レポートを Cisco UCS Director に追加し、列エントリを追加するテーブルを変更します。『Cisco UCS Director Orchestration Guide』の「Using Script Modules」の章を参照してください。

## コンポーネント

スクリプトモジュール

## コード

表形式レポートに次のコードを追加します。

```
{
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);
}
```

**結果**

表形式レポートがカスタム タスクからアクセスされ、要件が実行されます。

**実装**

カスタム タスクから表形式レポートにアクセスするには、カスタム タスクを作成し、変数を定義します。変数名はスクリプト モジュールで作成した表形式レポートと同じである必要があります。