Cisco UCS Director Cloupia スクリプト説明書

リリース 1.4
発行：2014 年 7 月
改訂：2015 年 2 月

Cisco Systems, Inc.
www.cisco.com

シスコは世界各国 200 箇所にオフィスを開設しています。
各オフィスの住所、電話番号、FAX 番号は次のシスコ Web サイトをご覧ください。
www.cisco.com/go/offices

このマニュアルに記載されている仕様および製品に関する情報は、予告なしに変更されることがあります。ALL このマニュアルに記載されている表現、情報、および推奨事項は、すべて正確であると考えていますが、明示的であれ黙示的であれ、一切の保証の責任を負わないものとします。このマニュアルに記載されている製品の使用は、すべてユーザ側の責任になります。

対象製品のソフトウェア ライセンスおよび限定保証は、
製品に添付された『Information Packet』に記載されています。添付されていない場合には、代理店にご連絡ください。

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

ここに記載されている他のいかなる保証にもよらず、各社のすべてのマニュアルおよびソフトウェアは、障害も含めて「現状のまま」として提供されます。シスコおよびこれら各社は、商品性の保証、特定目的への準拠の保証、および権利を侵害しないことに関する保証、あるいは取引過程、使用、取引慣行によって発生する保証をはじめとする、明示されたまたは黙示された一切の保証の責任を負わないものとします。

いかなる場合においても、シスコおよびその供給者は、このマニュアルの使用または使用できないことによって発生する利益の損失やデータの損傷をはじめとする、間接的、派生的、偶発的、あるいは特殊な損害について、あらゆる可能性がシスコまたはその供給者に知らされていても、それらに対する責任を一切負わないものとします。

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. シスコの商標の一覧は、
URL: www.cisco.com/go/trademarks. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

このマニュアルで使用している IP アドレスは、実際のアドレスを示すものではありません。マニュアル内の例、コマンド出力、およびその他の図は、説明のみを目的として使用されています。説明の中に実際のアドレスが使用されていたとしても、それは意図的なものではなく、偶然の一致によるものです。

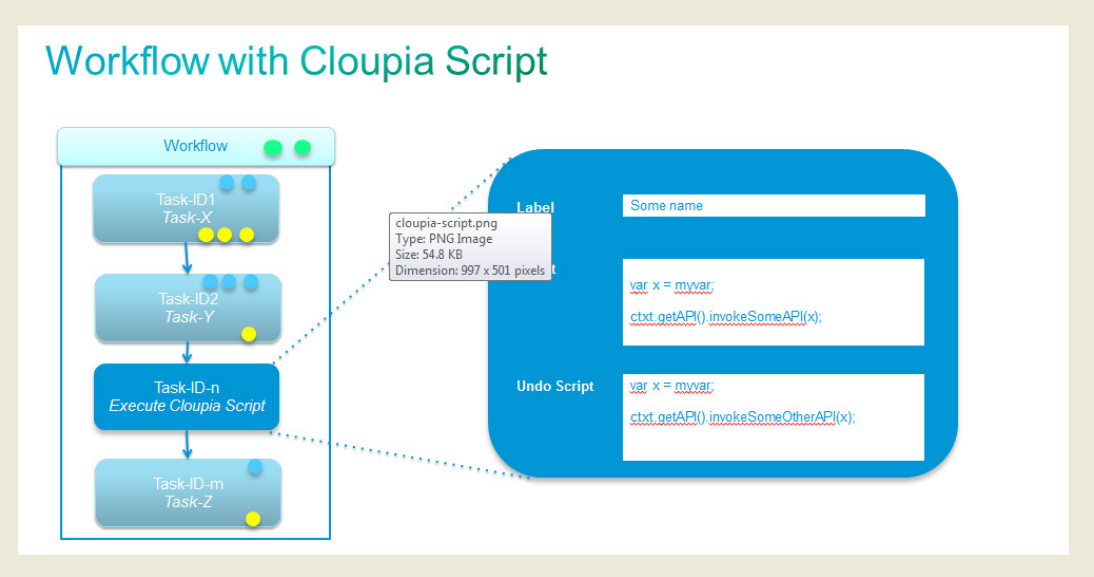
© 2014 - 2015 Cisco Systems, Inc. All rights reserved.

目次

1 使用する前に 4
2 CloupiaScript を使用したロギング 5
3 カスタム タスク 6
3.1 新しいカスタム タスクの追加 6
3.1.1 カスタム タスク入力の定義 6
3.1.2 カスタム タスク出力の定義 7
3.1.3 カスタム タスク スクリプトの作成 7
3.2 カスタム タスクのエクスポート 8
3.3 カスタム タスクのインポート 9
4 CloupiaScript での数字の処理 11
4.1 カスタム タスクでの入力タイプの定義 12
5 ネットワーク デバイスへのアクセス 13
6 ネットワーク要素アカウントへのアクセス 14
7 仮想アカウントと物理アカウントのセッションまたは API へのアクセス 15
8 レポートへのアクセス 19
9 レポートの電子メール送信 20
10 デリゲート API へのアクセス 21
11 データベース テーブルのアクセスと操作 21
12 カタログ プロビジョニングの繰り返し 22
13 サービス要求の入力または出力の取得 23
14 ワークフロー タスクの呼び出しまたは実行 24
15 タスクの成功または失敗状態への設定 25
16 その他のワークフローの呼び出し 25
17 機能クラスへのアクセス 26
18 リモート Cisco UCS Director 上での API の呼び出し 26
19 クリティカル セクション 27
20 ポッド全体での VLAN の完全なリストの取得 28
21 ポッド単位またはポッド全体で使用される IP の完全なリストの取得 28
22 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 30
23 vDC 経由の複数 VM の移動 30
24 ワークフロー内のタスクのロールバック 31

1 使用する前に

CloupiaScript は JavaScript と Cloupia ライブラリを組み合わせたもので、Cloupia サーバ プラットフォームから実行できます。



ハイライト

- スクリプトから Cloupia Java API とオブジェクトのサブセットにアクセスできます。
- CoupiaScript はすべての Java スクリプト構文をサポートします。
- CloupiaScript にはサーバ側のスクリプト記述が含まれているため、CoupiaScript はブラウザ ベースのオブジェクトをサポートしません。

スクリプト コンテキスト変数

CloupiaScript で事前に定義されている 3 つの最上位変数は次のとおりです。

- ctxt
- logger
- util

オブジェクト:ctxt

この変数は、Cloupia ワークフロー実行コンテキストを指します。スクリプトはこのオブジェクトを使用して現在のワークフロー、タスク、入力、出力の情報にアクセスできます。また、Cloupia API にもアクセスできます。

オブジェクト:logger

この変数は、Cloupia ワークフロー ロガー オブジェクトを指します。サービス リクエスト ログに記録される相当量のログ ステートメントをすべてのスクリプトが追加します。

オブジェクト:util

この変数を使用して、いくつかの組み込みユーティリティ メソッドにアクセスできます。

CloupiaScript の定義と実行

1. ワークフローを定義します。
2. タスク ライブラリで **Execute CloupiaScript** タスクを探して、ワークフロー デザイナにドラッグします。
3. CloupiaScript を記述します。
4. [実行] をクリックして CloupiaScript を実行します。

メッセージを出力するサンプル スクリプト:

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

上記スクリプトは「Hello World」というメッセージをログに出力します。

2 つの数値の合計を求めるサンプル スクリプト

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

2 CloupiaScript を使用したロギング

CloupiaScript には、サービス要求ログにメッセージを記録するために使用可能な API が用意されています。
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。

例:

```
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 カスタム タスク

Cisco UCS Director：

- 入力と出力が明確に定義された新しいワークフロー タスクを追加できるようにします。
- スクリプトを使用してロジックを実装します。
- CloupiaScript をサポートします。
- システム間のカスタム タスクのエクスポートとインポートを有効にします。
- 次のパスに沿ってカスタム タスクを管理できるようにします。
  [ポリシー] → [オーケストレーション] → [カスタムタスク]

カスタム タスクが作成されたら、組み込みタスクとカスタム タスクの動作に本質的な違いはありません。カスタム タスクは組み込みタスクや他のカスタム タスクを内部的に呼び出すことができます。また、カスタム タスクは他のタスクのラッパーとして機能します。

3.1 新しいカスタム タスクの追加

新しいカスタム タスクを定義するには、次の手順を実行します。

1. タスクの基本的な詳細（タスク名、ラベル、要約、および説明）を指定します。
2. タスクを分類すべきカテゴリを指定します。
3. タスクをオーケストレータに登録し、ワークフローですぐに使用できるようにする場合は、**Activate** フラグを有効にします。
4. タスクの入力を定義します。詳細については、カスタム タスクの入力を参照してください。
5. タスクの出力を定義します。詳細については、カスタム タスクの出力を参照してください。
6. スクリプトを作成します。詳細については、カスタム タスク スクリプトの作成を参照してください。

3.1.1 カスタム タスク入力の定義

- **入力フィールド名**：変数の内部名。先頭を英字にする必要があり、スペースや特殊文字を含めることはできません。
- **入力フィールド ラベル**：入力フィールドを識別するラベル。
- **入力フィールドタイプ**：入力カテゴリのタイプを選択します。カテゴリは、ユーザに値の入力を促す方法を定義したものです。
- **入力タイプにマッピング**：（オプション）このタスク入力をワークフロー ユーザ入力にマップ可能にする必要がある場合は、この変数にマップ可能な変数のタイプを指定します。

- **入力フィールドヘルプ**：（オプション）フィールド上にマウス カーソルを移動したときに表示される説明。
- **入力フィールド注記**：（オプション）入力フィールドのヒント テキスト。
- **フィールドグループ名**：指定した場合は、グループが一致するすべてのフィールドがフィールド グループに配置されます。
- **必須**：オンにした場合は、ユーザがこのフィールドの値を指定する必要があります。

✐ 入力フィールド タイプに基づいて、フィールド属性を定義する必要があります。

- テキスト フィールド属性は、入力フィールド タイプが **Text** の場合に適用されます。
- LOV フィールド属性は、入力フィールド タイプが **LOV（値のリスト）またはラジオボタン付きの LOV** の場合に適用されます。
- テーブル名は、入力フィールド タイプが**テーブルやポップアップ テーブル、または選択チェックボックス付きのテーブル**の場合に適用されます。

3.1.2 カスタム タスク出力の定義

- **出力フィールド名**：出力フィールドの一意の名前。先頭を英字にする必要があり、スペースや特殊文字を含めることはできません
- **出力フィールドの説明**：出力フィールドの説明。
- **出力フィールドのタイプ**：出力のタイプ。出力を他のタスク入力にマップする方法を決定します。

✐ 内部的には、出力のタイプに関係なく、すべての出力が文字列オブジェクトとして扱われます。

3.1.3 カスタム タスク スクリプトの作成

CloupiaScript はカスタム タスクのハンドラを定義するために使用されます。CloupiaScript では 6 つの最上位変数（ctxt、logger、util、input、output、および metadata）が事前に定義されています。

- **オブジェクト ctxt**：この変数は Cloupia ワークフロー実行コンテキストを指します。このオブジェクトを使用して、スクリプトは、現在のワークフロー、タスク、入力、および出力に関する情報にアクセスできるほか、Cloupia API にもアクセスできます。Cloupia API を使用して作成/読み取り/更新/削除（CRUD）操作の呼び出し、ワークフロー タスクの呼び出し、他の REST API の公開ができます。
- **オブジェクト logger**：この変数またはオブジェクトは Cloupia ワークフロー ロガー オブジェクトを指します。サービス リクエスト ログに記録される相当量のログ ステートメントをすべてのスクリプトが追加します。
- **オブジェクト util**：この変数を使用して、いくつかの組み込みユーティリティ メソッドにアクセスできます。

- **オブジェクト input**：この変数はすべてのタスク入力変数を保持している単純な POJO（オブジェクト）を指します。
- **オブジェクト output**：この変数はすべてのタスク出力変数を保持している単純な POJO（オブジェクト）を指します。
- **オブジェクト metadata**：この変数はタスク定義オブジェクトを指しており、実装スクリプトでは使用されません。

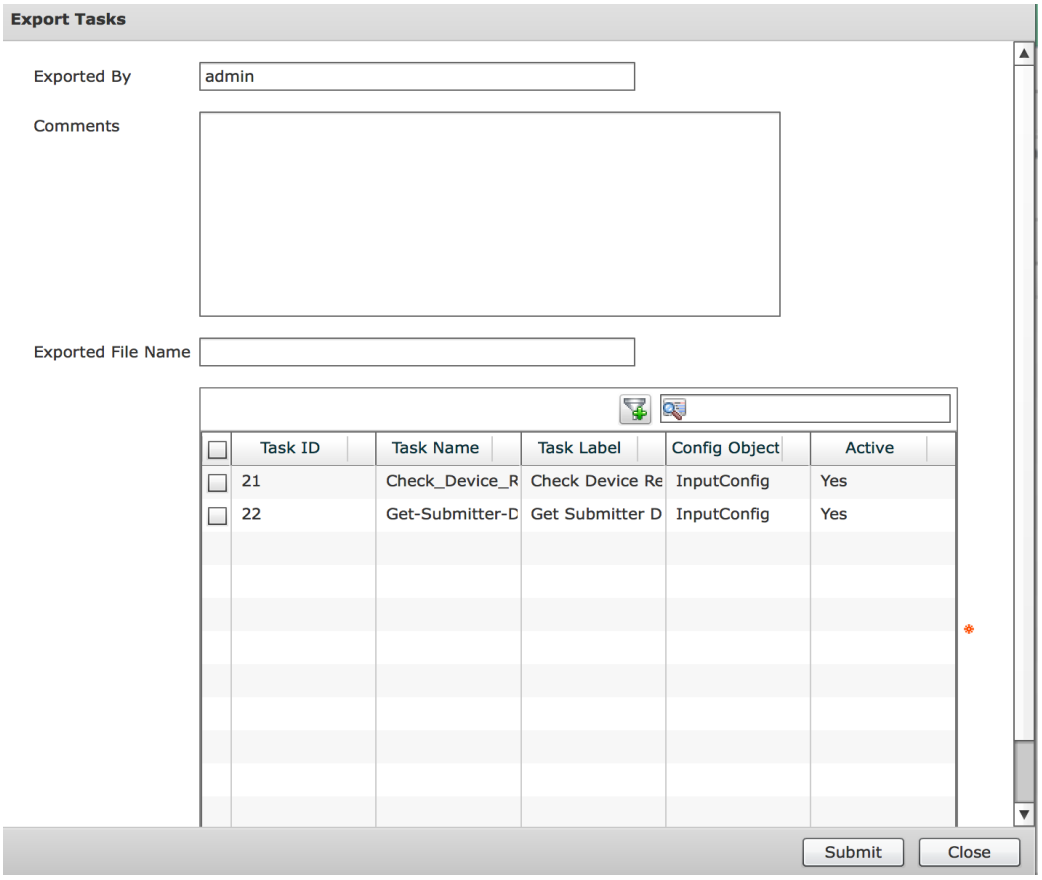
> カスタム タスクで ctxt.setOutputValue() と ctxt.updateInput() が使用されていないことを確認してください。

3.2 カスタム タスクのエクスポート

ローカル システムにカスタム ワークフロー タスクをエクスポートして、タスクを更新または変更できます。その後で、更新したカスタム ワークフロー タスクを Cisco UCS Director にインポートできます。

カスタム タスクをエクスポートするには、次の手順を実行します。

1. [ポリシー] > [オーケストレーション] に移動して、[カスタムワークフロータスク] タブをクリックします。
2. エクスポートするカスタム タスクを選択します。
3. [タスクのエクスポート] をクリックします。
   [タスクのエクスポート] ダイアログボックスが表示されます。



[タスクのエクスポート] ダイアログボックスで、次のフィールドに値を入力します。

- [エクスポート実行ユーザ]：エクスポートするユーザの Cisco UCS Director ユーザ ID。
- [コメント]：エクスポートするタスクまたはタスクのグループに関する注記。
- [エクスポートされたファイルの名前]：システムがファイル名を自動生成します。または、エクスポートしたカスタム ワークフロー タスク ファイルのファイル名を指定できます。

4. エクスポートするタスクのリストから 1 つ以上のタスクを選択します。
5. [送信] をクリックします。

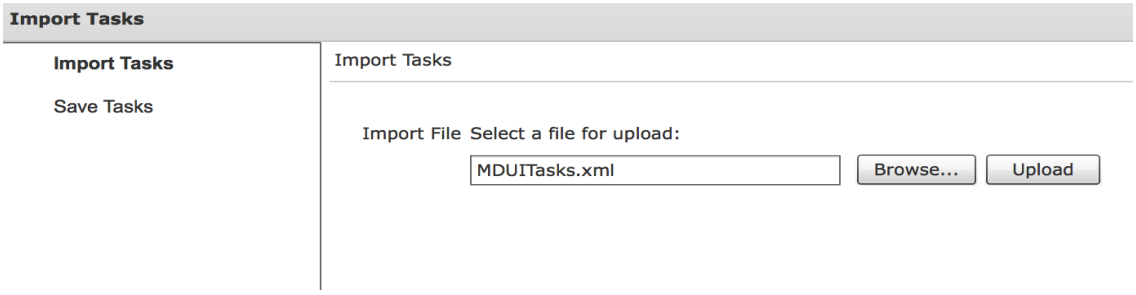
タスクのエクスポート アクションによって、システムにインポート可能な拡張子 .wfdx 付きのファイルが作成されます。

3.3 カスタム タスクのインポート

カスタム ワークフロー タスクを更新したら、システムから Cisco UCS Director にカスタム ワークフロー タスクをインポートできます。

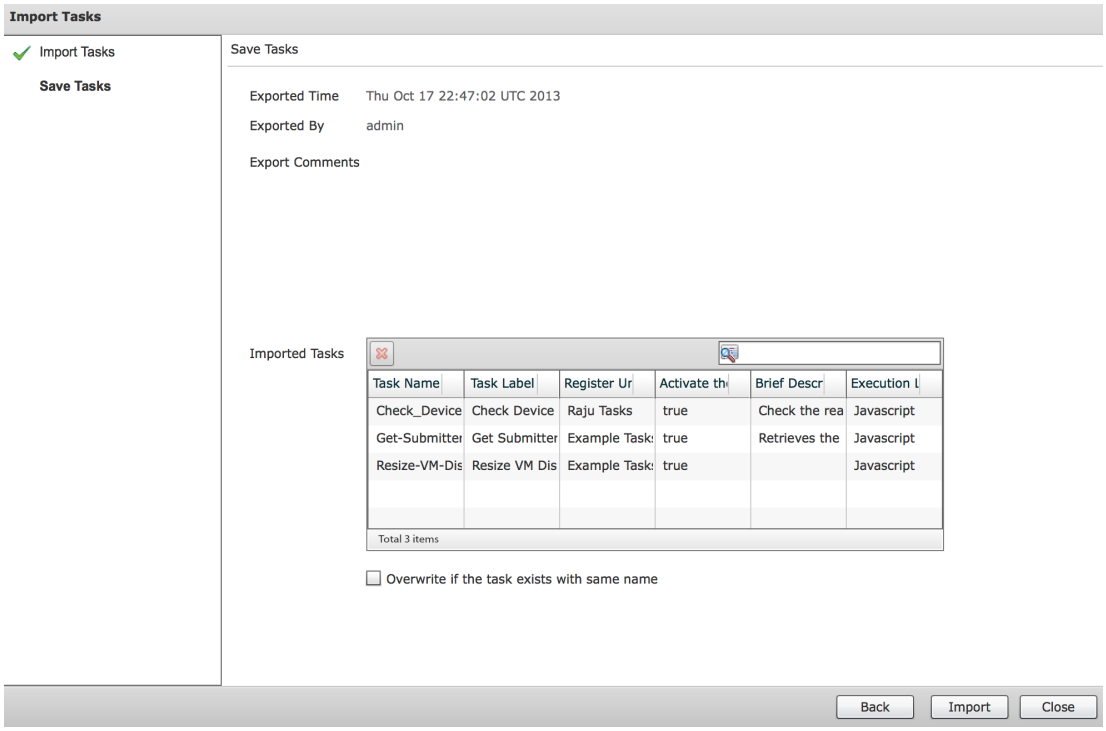
カスタム タスクをインポートするには、次の手順を実行します。

1. [ポリシー] > [オーケストレーション] に移動して、[カスタムワークフロータスク] タブをクリックします。
2. インポートするカスタム タスクを選択します。
3. [タスクのインポート] をクリックします。
   [タスクのインポート] ダイアログボックスが表示されます。



4. [参照] をクリックして、アップロードするエクスポートしたファイルを選択し、[アップロード] をクリックします。

5. [タスクの保存] をクリックします。これにより、タスクをエクスポートした時刻、タスクをエクスポートしたユーザ名、およびタスクのエクスポート時に指定されたコメントが表示されます。インポートしたタスクの詳細が表形式で表示されます。



6. インポートしたタスクで既存のタスクを上書きするには、[同じ名前のタスクが存在する場合は上書き] チェックボックスをオンにします。

   ✐ [同じ名前のタスクが存在する場合は上書き] チェックボックスがオフになっている場合は、新しいタスクが、名前を一意にする番号（n）付きで追加されます。

7. [インポート] をクリックします。

カスタム タスクの例：送信者の詳細を取得する

```
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();
```

詳細:

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)

内部タスク

- 内部タスクが現在のサービス要求(SR)に対するログ メッセージを生成します。
- 内部タスクが現在の SR にオブジェクト変更を登録します。
- 内部タスクが現在の SR に取り消しタスクを登録します(そのため、ロールバックが正しく機能します)。
- 内部タスクの出力はカスタム タスクの外部で自動的に表示されません。スクリプト開発者が "output.X = Y" 式を使用して、その出力を明示的に保存する必要があります。
- 各カスタム タスクは複数の内部タスクを呼び出すことができます。
- 各カスタム タスクはループで内部タスクを呼び出す場合があります。
- 内部タスクが失敗した場合は、外部カスタム タスクが自動的に失敗します。
- 内部タスクは "try { } catch(e) { }" ブロックを使用して変更できます。

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

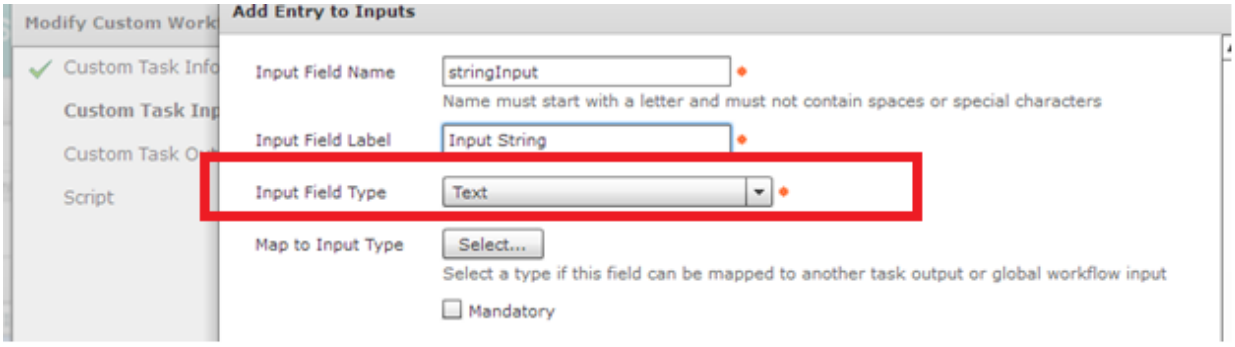
4 CloupiaScript での数字の処理

次の方法のいずれかで CloupiaScript 内の数字入力を処理できます。

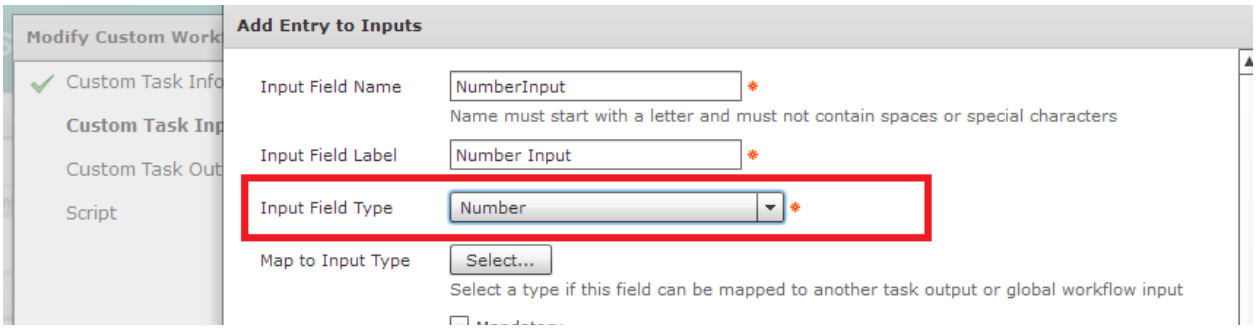
1. 文字列変数を整数に変換する。
2. int 変数を整数に変換する。

4.1 カスタム タスクでの入力タイプの定義

1. カスタム タスク入力をテキストとして定義する



2. カスタム タスク入力を数字として定義する



入力をテキストと数字として定義するための CloupiaScript スニペットを以下に示します。

1. テキストとして定義された入力

```
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();
```

この例では、カスタム タスクのユーザ入力が求められます。入力がヌルでない場合は、入力文字列が数字に変換され、ログに出力されます。

特定の入力が数字でない場合は、NumberFormatException がログにスローされます。

2. 数字として定義された入力

```
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 ネットワーク デバイスへのアクセス

ネットワーク デバイスは SSH を使用してアクセスできます。

ネットワーク デバイスにアクセスするには、次の手順を実行します。

1. デバイスの詳細を渡すことによって、SSHClient（com.cloupia.lib.util.ssh.SSHClient）インスタンスを作成します。
2. 接続してセッションを開きます。
3. PrintStream オブジェクトを使用してデバイスと通信します。
4. クライアントを切断します。

次のスニペットは、デバイス上でのコマンドの発行方法を示しています。

タスクの入力:

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

サンプル スクリプト:

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

```
        var client =  new SSHClient(input.ipAddress, 22,
        input.userName, input.password);
        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 ネットワーク要素アカウントへのアクセス

次のスニペットは、ネットワーク要素のアカウントへのアクセス方法を示しています。

サンプル スクリプト:

```
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 仮想アカウントと物理アカウントのセッションまたは API へのアクセス

機能は Cisco UCS Director 機能で定義されます。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。セッション オブジェクトも Cisco UCS Director で定義されます。必要なセッション オブジェクトを作成し、API にアクセスします。

NetApp

次のスニペットは、Netapp LUN マッピング機能を示しています。

```
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

```
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();
```

vcenterDriver API を使用して、VMware API と CRUD 操作を呼び出します。

例:

```
driver.updateSingleVM(Account creds, GenericVM gvm, VMWareVMSummary
vmsummary, boolean includeSnapshotInfo);
```

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

REST API を使用した場合：

```
APIProvider.getInstance().performAction(ReportContext context,
VMActionUtil.ACTION_DELETE_VM_DISKS, String comment, String userId,
ActionParam[] actionParams)
```

インベントリの呼び出し：

```
var controller =
com.cloupia.service.cIM.inframgr.InfraMgrImpl.getController(account.getA
ccountName());
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.gethPSAgentIP());
       var remoteAgent = new RemoteAgent(agent.getAddress(),
       agent.getPortNumber(), agent.getAccessKey());
       var targetServer = new TargetServer(creds.getHServer(),
       creds.gethDomain() + "\\" + creds.getHUserId(),
       creds.getHPasswd());
       var api = new SCVMMAPI(remoteAgent,
       targetServer,SCVMMAPI.TIME_5_MINS);
       logger.addInfo("Hyperv api: "+api);

}
accessHyperVAccount();
```

サンプル 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{
       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,I
	nfraAccountTypes.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.getAccount
	Name(),InfraAccountTypes.EMC_VMAX);
	var vmaxapi = new
	com.cloupia.lib.cIaaS.emc.vmax.EmcVmaxApi();
```

EMC VMAX タスクに対して Vmax API を使用します。

```
	var account =
	com.cloupia.service.cIM.inframgr.InfrastructureDataUtil.get
	AccountByType(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);
}
```

例:

```
api.sendCimRequest("CreateRAIDGroupWithPowerSavingSetting"
```

Use the VNX API for the EMC VNX tasks/operations.

インベントリ:

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

InventoryManager.getBlockFullInventory(account);
```

Whiptail

```
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.getAccount(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.getAccountName(),"lun",
      WTLUNs.class);
}
```

Cisco UCS Manager アカウント接続

Cisco UCS を設定または管理するには、Cisco UCS Manager アカウント接続を確立する必要があります。

次のサンプル コードは、Cisco UCS Manager アカウント接続を確立するためのものです。

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

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

accessUCSMAccount();
```

フィールドの説明:

1. <Account Name>:Cisco UCS Manager アカウントの名前。
2. InfraAccountTypes.UCSM:アカウントのタイプ。

8 レポートへのアクセス

CloupiaScript を使用して、既存の Cisco UCS Director レポートにアクセスし、レポート データから動的決定を下すことができます。
次のスクリプトは、レポートにアクセスしてレポート データを取得する方法を示しています。

```
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 coloumns
//returns vms those Power State coloumn value is ON
report = report.filterRowsByColumn("Power State", "ON", false);
logger.addInfo("report "+report.getReport);
}

accessReports();
```

9 レポートの電子メール送信

Cisco USC Director レポートをユーザに電子メール送信できます。レポートは、電子メール ワークフロー用のワークフロー スケジューラにより必要な頻度で定期的に電子メール送信できます。

次の例は、電源がオンになっているすべての VM のリストをワークフロー入力変数（Email Address）で指定されたユーザに電子メール送信します。

```
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 デリゲート API へのアクセス

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 にアクセスできます。このメソッドはデリゲート オブジェクトを返します。

コード例:
サンプル コード内で、デリゲート（UserAPIChargeBack）の名前空間が強調表示されています。

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

デリゲートを取得したら、デリゲート クラス内の任意のメソッドを実行できます。

11 データベース テーブルのアクセスと操作

次のコード例に示すように、データベース内のテーブルを表す対応する PoJo を getStore() メソッドに渡すことによって、Cisco USC Director データベースのテーブルにアクセスして操作できます。

コード例:
サンプル コード内で、データベース内の UCS サーバ テーブルに関連する PoJo クラスが強調表示されています。

```
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());
```

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。

12 カタログ プロビジョニングの繰り返し

カタログをプロビジョニングするためには次の入力が必要です。

- CATALOG_ID　　:プロビジョニングするカタログの ID。
- VDC_ID　　:特定のカタログの VDC の ID。
- PROVISION_QTY :整数値としてのプロビジョニングする必要のある回数。

```
var catId = ctxt.getInput(CATALOG_ID);
var vdcId = ctxt.getInput(VDC_ID);
var quantity = Integer.valueOf ( ctxt.getInput(PROVISION_QTY));
```

その入力から、次のようにオブジェクトをロードする必要があります。

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

カタログを n 回プロビジョニングするためのループを作成し、スレッド スリープメソッドを使用してプロビジョニング間に遅延を挿入します。

実行するための追加のパラメータを作成する:

ステータスを表示するためのサービス要求 ID の配列。

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

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;
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);
}
```

次のコードを使用して API の実行が成功したかどうかをチェックします。

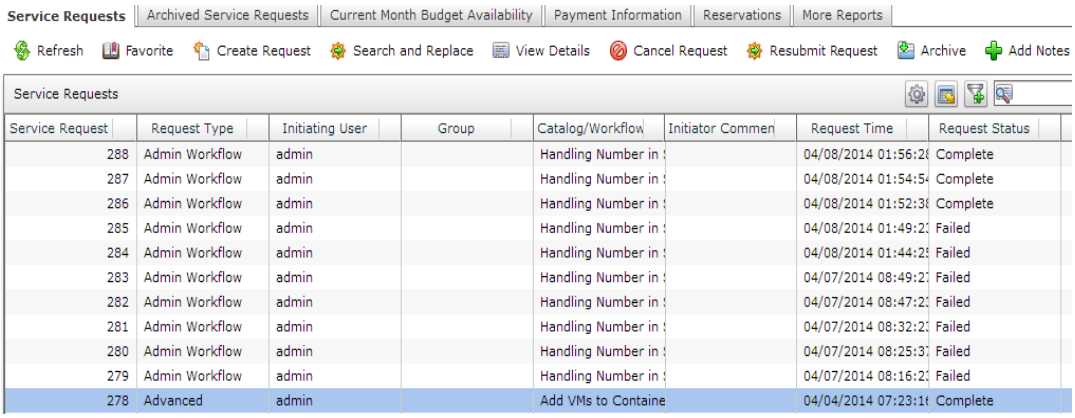
```
for (var i=0; i<childSrIdArray.length; i++)HHH
{
var childSrId = childSrIdArray[i];
var status = ctxt.waitForCompletion(childSrId, 1800000);
```

```
if (status == 0)
{
logger.addInfo("Provisioned SR ID  ="+ childSrId+ "
successfully.");
} else {
logger.addError("SR ID  ="+ childSrId+ " failed");
}
}
```

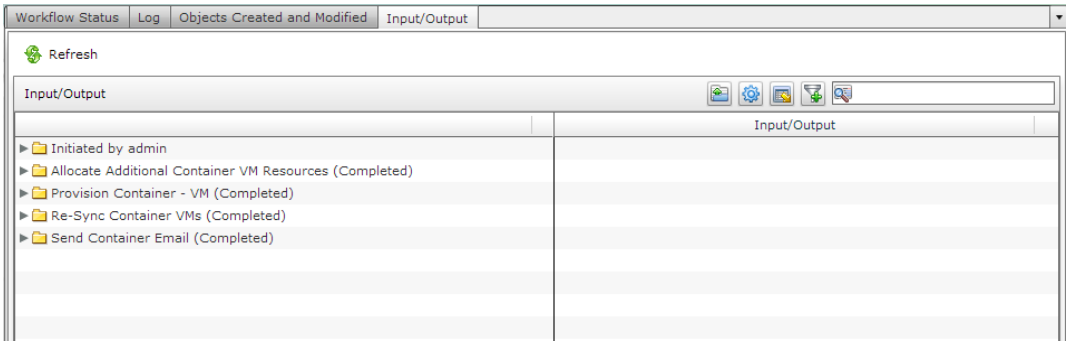
13 サービス要求の入力または出力の取得

サービス要求の入力または出力を表示するには、次の手順を実行します。

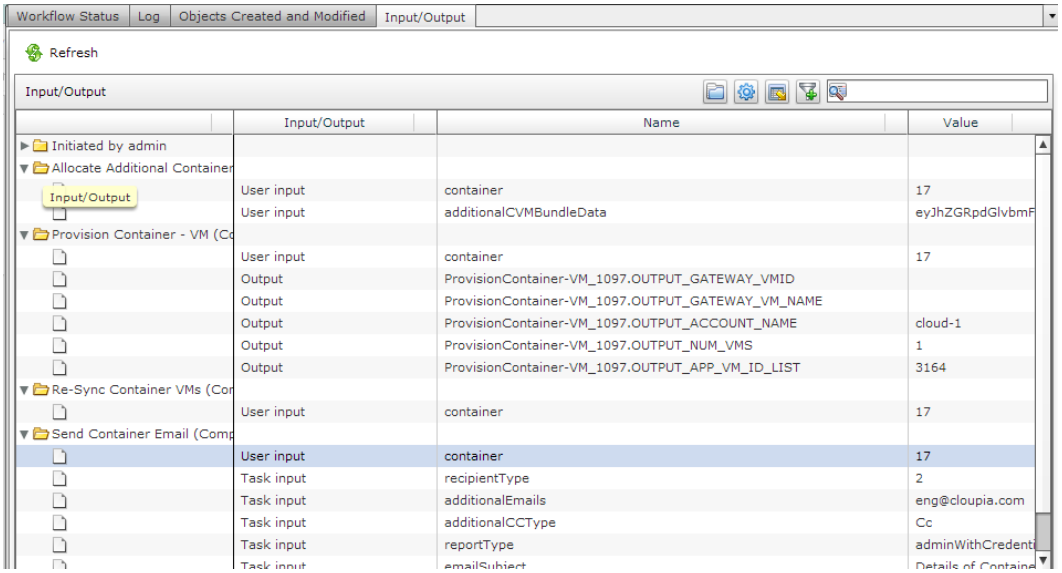
1. [組織] > [サービスリクエスト] に移動して、[サービスリクエスト] タブをクリックします。



2. サービス要求を選択して、[詳細の表示] をクリックします。
   ポップアップ ウィンドウが表示されます。
3. [入力/出力] をクリックします。



ワークフローの入力と出力、およびサブ ワークフローが表示されます。



14 ワークフロー タスクの呼び出しまたは実行

内部タスクを使用して CloupiaScript からワークフロー タスクを呼び出すことができます。

次のコード スニペットは、CloupiaScript の **ctxt** 暗黙的オブジェクトを使用してワークフロー タスクを呼び出す方法を示しています。

```
// 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);
```

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

}

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

15 タスクの成功または失敗状態への設定

次のスニペットは、タスクを成功または失敗に設定する方法を示しています。

```
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 その他のワークフローの呼び出し

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。

- **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。

次の例は、Task_X から Task_Y を呼び出す方法を示しています。

```
var params = util.createNameValueList();
params.addNameValue(util.createNameValue("Input_A", "Value_A"));
params.addNameValue(util.createNameValue("Input_B", "Value_B"));
params.addNameValue(util.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);
```

✐ パラメータリストで指定する 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。

17 機能クラスへのアクセス

FeatureContainer を使用してクラスをロードする必要があります。

例:

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

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

API ブラウザで必要なサーバ IP アドレスを選択することによって、リモート Cisco USC Director 上で API を呼び出すことができます。

```
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.
```

例:

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[] {catalogName, 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
```

19 クリティカル セクション

クリティカル セクションは、1 回に 1 つのスレッドだけがロック メカニズムを使用してクリティカル セクションの入力と実行を行うことができる、同期されたブロックに似ています。

クリティカル セクション API を以下に示します。

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

コンテキスト srId で動作するワークフロー スレッドが sectionName または lockName が指定された特定のブロックまたはセクションに排他的アクセスする場合は、クリティカル セクションへの排他的アクセスを取得するためにクリティカル セクション API が使用されます。他のスレッドがクリティカル セクションに入る場合は、ロックを保持しているスレッドが exitCriticalSection API を使用してそれを解放するまで待機する必要があります。

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, CustomActionLogger logger) throws Exception
```

20 ポッド全体での VLAN の完全なリストの取得

入力 : ポッドまたはデータ センターまたはクラウド
出力 : POD1 --> v1、v2、v10
POD2 -> v11、v13 など
Total vLANS Used = 20.
Pools (vP1, vP2).

サンプル スクリプト :

✎ UCS クラウドが実装されます。

```
var accountName = input.UCSCLOUD;
var ucsVlanList = UcsDataPersistenceUtil.getVLAN(accountName);
var vLanList = new ArrayList();
for(var i=0;i<ucsVlanList.size();i++){
var ucsVLAN = ucsVlanList.get(i);
vLanList.add(ucsVLAN.getId());
}
output.UCSvLANList =  vLanList;
output.UCSvLANsUsed =  vLanList.size();
```

21 ポッド単位またはポッド全体で使用される IP の完全なリストの取得

入力 : ポッド名
出力 : そのポッド上のすべての IP のカンマ区切りリスト

サンプル スクリプト :

```
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){
        }else if(acc.getAccountType()==Account.VMWARE){
                listOfIP.add(acc.getVServer());
        }else if(acc.getAccountType()==Account.HYPERV){
                listOfIP.add(acc.getHServer());
            }else if(acc.getAccountType()==Account.XENDESKTOP){
                listOfIP.add(acc.getXenCtrlIP());
```

```
                        }
                }

        output.IPList = listOfIP;
      }
    getListOfIPs();
```

サンプル スクリプト

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

```
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 Execption 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 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

入力 : VMware アカウント、VMware クラスタ名

出力 : データストア タスクとして Associate VNX LUN にマップ可能なホストのカンマ区切りリスト ただし、この操作はクラスタ内のすべてのホストに適用され、1 つのホストだけには適用されません。

サンプル スクリプト:

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

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

23 vDC 経由の複数 VM の移動

入力 : VDC-one、VDC-two

出力 : VDC-one 内のすべての VM が VDC-two に移動

サンプル スクリプト:

```
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.getVMDataByVMId(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){
logger.addError("Error occured 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 ワークフロー内のタスクのロールバック

ワークフロー内のタスクのロールバック/取り消し

ワークフロー内のタスクをロールバックする場合は、change tracker API を使用してタスクのロールバックを処理します。

スクリプト例:

```
importPackage(com.cloupia.service.cIM.inframgr.customaction
s);
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();
```

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 も同じパラメータのセットを使用して呼び出します。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 change tracker API を使用した取り消しサポートが必要です。