Cisco UCS Director リリース 6.0 カスタム タスク スタートアップ ガイド

初版：2016 年 09 月 16 日

シスコシステムズ合同会社
〒107-6227 東京都港区赤坂9-7-1 ミッドタウン・タワー
http://www.cisco.com/jp
お問い合わせ先：シスコ コンタクトセンター
0120-092-255 （フリーコール、携帯・PHS含む）
電話受付時間：平日 10:00～12:00、13:00～17:00
http://www.cisco.com/jp/go/contactcenter/

【注意】シスコ製品をご使用になる前に、安全上の注意（**www.cisco.com/jp/go/safety_warning/**）をご確認ください。本書は、米国シスコ発行ドキュメントの参考和訳です。リンク情報につきましては、日本語版掲載時点で、英語版にアップデートがあり、リンク先のページが移動/変更されている場合がありますことをご了承ください。あくまでも参考和訳となりますので、正式な内容については米国サイトのドキュメントを参照ください。また、契約等の記述については、弊社販売パートナー、または、弊社担当者にご確認ください。

目 次

はじめに

対象読者

このマニュアルは、Cisco UCS Director を使用し、以下の少なくとも 1 つの分野において責任と専門知識を持つデータセンター管理者を主に対象としています。

- サーバ管理
- ストレージ管理
- ネットワーク管理
- ネットワーク セキュリティ
- 仮想化および仮想マシン

表記法

テキストのタイプ	表示
GUI 要素	タブの見出し、領域名、フィールドのラベルのようなGUI要素は、[GUI 要素（this font）] のように示しています。 ウィンドウ、ダイアログボックス、ウィザードのタイトルのようなメイン タイトルは、[メイン タイトル（this font）] のように示しています。

テキストのタイプ	表示
マニュアルのタイトル	マニュアルのタイトルは、イタリック体（*italic*）で示しています。
TUI 要素	テキストベースのユーザ インターフェイスでは、システムによって表示されるテキストは、courier フォントで示しています。
システム出力	システムが表示するターミナル セッションおよび情報は、courier フォントで示しています。
CLI コマンド	CLI コマンドのキーワードは、ボールド体（**bold**）で示しています。 CLI コマンド内の変数は、イタリック体（*italic*）で示しています。
[ ]	角カッコの中の要素は、省略可能です。
{x | y | z}	どれか1つを選択しなければならない必須キーワードは、波カッコで囲み、縦棒で区切って示しています。
[x | y | z]	どれか1つを選択できる省略可能なキーワードは、角カッコで囲み、縦棒で区切って示しています。
string	引用符を付けない一組の文字。stringの前後には引用符を使用しません。引用符を使用すると、その引用符も含めて string とみなされます。
< >	パスワードのように出力されない文字は、山カッコで囲んで示しています。
[ ]	システム プロンプトに対するデフォルトの応答は、角カッコで囲んで示しています。
!、#	コードの先頭に感嘆符（!）またはポンド記号（#）がある場合には、コメント行であることを示します。

（注） 「注釈」です。役立つ情報や、このマニュアル以外の参照資料などを紹介しています。

注意 「要注意」の意味です。機器の損傷またはデータ損失を予防するための注意事項が記述されています。

ヒント　「問題解決に役立つ情報」です。ヒントには、トラブルシューティングや操作方法ではなく、ワンポイントアドバイスと同様に知っておくと役立つ情報が記述される場合もあります。

ワンポイント アドバイス　「時間の節約に役立つ操作」です。ここに紹介している方法で作業を行うと、時間を短縮できます。

警告　安全上の重要事項

「危険」の意味です。人身事故を予防するための注意事項が記述されています。機器の取り扱い作業を行うときは、電気回路の危険性に注意し、一般的な事故防止対策に留意してください。各警告の最後に記載されているステートメント番号を基に、装置に付属の安全についての警告を参照してください。

これらの注意事項を保管しておいてください。

関連資料

『Cisco UCS DirectorDocumentation Roadmap』

Cisco UCS Director の資料の詳細なリストについては、次の URL にある 『*Cisco UCS Director Documentation Roadmap*』 を参照してください。http://www.cisco.com/en/US/docs/unified_computing/ucs/ucs-director/doc-roadmap/b_UCSDirectorDocRoadmap.html

『Cisco UCS Documentation Roadmaps』

すべての B シリーズ マニュアルの一覧については、『*Cisco UCS B-Series Servers Documentation Roadmap*』（URL：http://www.cisco.com/go/unifiedcomputing/b-series-doc）を参照してください。

すべての C シリーズ マニュアルの一覧については、http://www.cisco.com/go/unifiedcomputing/c-series-doc で入手できる『*Cisco UCS C-Series Servers Documentation Roadmap*』を参照してください。

（注）　『*Cisco UCS B-Series Servers Documentation Roadmap*』には Cisco UCS Manager および Cisco UCS Central のドキュメントのリンクが含まれています。『*Cisco UCS C-Series Servers Documentation Roadmap*』には Cisco Integrated Management Controller のドキュメントのリンクが含まれています。

マニュアルに関するフィードバック

このマニュアルに関する技術的なフィードバック、または誤りや記載もれなどお気づきの点がございましたら、ucs-director-docfeedback@cisco.com までコメントをお送りください。ご協力をよろしくお願いいたします。

マニュアルの入手方法およびテクニカル サポート

マニュアルの入手、Cisco Bug Search Tool（BST）の使用、サービス要求の送信、追加情報の収集の詳細については、『What's New in Cisco Product Documentation』を参照してください。

新しく作成された、または改訂されたシスコのテクニカル コンテンツをお手元に直接送信するには、『What's New in Cisco Product Documentation』RSS フィードをご購読ください。RSS フィードは無料のサービスです。

第 1 章

このリリースの新規情報および変更情報

このリリースの新規情報および変更情報

このガイドでは、新しいリリースに関する大幅な変更は行われていません。

このリリースの新規情報および変更情報

第 2 章

カスタム タスクの概要

カスタム タスクを使用する理由

カスタム タスクは、Cisco UCS Director Orchestrator の機能を拡張します。カスタム タスクを使用することで、事前定義されているタスクや Cisco UCS Director に同梱されているワークフローで使用できない機能を作成することができます。カスタム タスク内からレポートの生成、物理リソースまたは仮想リソースの設定、および他のタスクの呼び出しを行うことができます。

カスタム タスクの仕組み

カスタム タスクは、作成されて Cisco UCS Director にインポートされると、Cisco UCS Director Orchestrator の他のタスクと同様に機能します。カスタム タスクを変更、インポート、およびエクスポートしたり、ワークフローに追加したりできます。

カスタム タスクの使用方法

Cisco UCS Director 内からカスタム タスクを記述、編集、およびテストします。カスタム タスクを記述するには、管理者権限が必要です。

オーケストレーション操作を可能にする Cisco UCS Director Java ライブラリを含む JavaScript のバージョンである CloupiaScript を使用して、カスタム タスクを記述します。そして、ワークフロー内のタスクを含む他のタスクのようなカスタム タスクを使用して、コンポーネント上で作業を調整します。

CloupiaScript はすべての JavaScript 構文をサポートします。また、CloupiaScript は Cisco UCS Director Java ライブラリのサブセットへのアクセスをサポートし、カスタム タスクが Cisco UCS Director コンポーネントにアクセスできるようにします。CloupiaScript はサーバでのみ動作するため、クライアント側のオブジェクトはサポートされていません。

CloupiaScript は Nashorn スクリプト エンジンを使用します。Nashorn の詳細については、https://docs.oracle.com/javase/8/docs/technotes/guides/scripting/nashorn/api.htm で Oracle の Web サイトにあるテクニカル ノートを参照してください。

カスタム タスクの暗黙的な変数

3 つの事前に定義された最上位レベルの変数がカスタム タスクに自動的に含まれています。

変数	説明
ctxt	ワークフロー実行コンテキスト。このコンテキスト オブジェクトには、現在のワークフロー、現在のタスク、および使用可能な入出力に関する情報が含まれています。また、Cisco UCS Director Java API へのアクセス権も含まれており、これを使用して作成、読み取り、更新、および削除（CRUD）の操作の実行、他のタスクの起動、他の API メソッドの呼び出しを行うことができます。*ctxt* 変数は、プラットフォーム API クラス `com.cloupia.service.cIM.inframgr.customactions` のインスタンスです。 `CustomActionTriggerContext`
logger	ワークフロー `logger` オブジェクト。ワークフロー ロガーは、サービス要求（SR）のログに書き込みを行います。*logger* 変数は、プラットフォーム API クラス `com.cloupia.service.cIM.inframgr.customactions.CustomActionLogger` のインスタンスです。
util	ユーティリティ メソッドへのアクセス権を提供するオブジェクト。*util* 変数は、プラットフォーム API クラス `com.cloupia.lib.util.managedreports.APIFunctions` のインスタンスです。

暗黙的な変数の API クラスの詳細については、Cisco UCS Director スクリプト バンドルに含まれている CloupiaScript Javadoc を参照してください。

JDK アップグレードによる CloupiaScript への変更

Cisco UCS Director リリース 5.4 以降では、JDK バージョンが 1.6 から 1.8 にアップグレードされています。JDK 1.6 バージョンが Rhino JavaScript エンジンをベースにしていたのに対し、JDK 1.8 バージョンには新しい Nashorn Javascript エンジンが付属しています。Nashorn JavaScript エンジンには、構文の変更とスクリプト内の特定の関数およびクラスの使用方法の変更があります。

以下は、Cisco UCS Director リリース 5.5 のカスタム タスクをスクリプト化する際に留意するべき変更点です。

- **オブジェクト プロパティの値を取得するため、オブジェクトをマップに変換**

  Cisco UCS Director リリース 5.3 までは、For ループを使用して、各オブジェクト（vminfo など）のプロパティ値を取得するには、次のコード スニペットを使用する必要があります。

```
importPackage(com.cloupia.service.cIM.inframgr);
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.lib.util);
importPackage(java.util);
importPackage(java.lang);
var vmSummary ="";
var vminfo = ctxt.getAPI().getVMwareVMInfo(306);//306 is vmId
for(var x in vminfo){
//escaping getter and setter methods
if(x.match(/get*/) == null && x.match(/set*/) == null && x.match(/jdo*/) == null &&
x.match(/is*/)
 == null && x.match(/hashCode/) == null && x.match(/equals/) == null)
{
vmSummary += x  +":"+ vminfo[x] + '#';
};
};
logger.addInfo("VMSUMMARY="+vmSummary);
```

  Cisco UCS Director リリース 5.4 以降では、ObjectToMap クラスの convertObjectToMap () メソッドを使用してオブジェクト（vminfo など）をマップに変換してから、For ループでオブジェクトを使用してオブジェクトの値を取得する必要があります。次のコードスニペットは、オブジェクトの各プロパティの値の取得方法を示しています。

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

var vmSummary = "";
var vminfo = ctxt.getAPI().getVMwareVMInfo(4);//4 is vmId
var vminfo = ObjectToMap.convertObjectToMap(vminfo);
for (var x in vminfo) {
vmSummary += x  +":"+ vminfo[x] + '#';
}
logger.addInfo("VMSUMMARY="+vmSummary);
```

  （注） ObjectToMap.convertObjectToMap(vminfo) クラスはオブジェクト（vminfo など）にプリミティブまたは文字列型のプロパティが含まれている場合にのみ使用できます。ベスト プラクティスは、getVmId() や getVmName() などの標準の getter メソッドを使用してオブジェクトの値を取得することです。

- **print( ) 関数の使用**

  `println( )` の代わりに `print( )` を使用します。

  （注） JDK 1.8 は、後方互換性のために `println( )` を引き続きサポートします。

- **class<T> パラメータをメソッドまたはコンストラクタに渡すための構文の変更**

Class<T>パラメータをメソッドまたはコンストラクタに渡すための構文が変更されました。JDK1.6 では、次の構文は有効でした。

```
var fml = new FormManagedList(PrivateCloudNetworkPolicyNICPortGroup);
```

しかし、JDK1.8 では、次のように、.class を付加して PrivateCloudNetworkPolicyNICPortGroup Java クラスを引数として渡す必要があります。

```
var fml = new FormManagedList(PrivateCloudNetworkPolicyNICPortGroup.class);
```

- **クラスおよびパッケージをインポートする構文の変更**

クラスとパッケージをインポートする構文が変更されました。以前のインポート ステートメントではクラスまたはパッケージが javascript 実行のグローバル スペースで使用できるようになっており、それが必ずしも必要ではなかったのに対し、新しいインポート ステートメントではクラスまたはパッケージの使用のローカリゼーションが向上します。

ステートメントを Rhino JavaScript エンジンにインポートします。

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

ステートメントを Nashorn JavaScript エンジンにインポートします。

```
var CollectionsAndFiles = new JavaImporter( java.util, java.io, java.nio);
with (CollectionsAndFiles) {
 var files = new LinkedHashSet();
 files.add(new File("Filename1"));
 files.add(new File("Filename2"));
}
```

「with」ステートメントは、オブジェクトがメモリにロードされる時間についての引数として与えられた変数の範囲を定義します。たとえば、一度に多くの Java パッケージをインポートするのにも役立ちます。JavaImporter クラスを「with」ステートメントとともに使用できます。インポートされたパッケージからのすべてのクラス ファイルは、「with」ステートメントのローカル範囲内にアクセスできます。

Java パッケージのインポート：

```
var imports = new JavaImporter(java.io, java.lang);
with (imports) {
    var file = new File(__FILE__);
    System.out.println(file.getAbsolutePath());
    // /path/to/my/script.js
```

（注） 古い importPackage() および importClass() ステートメントは、後方互換性のために Cisco UCS Director 5.5 で引き続きサポートされます。カスタム タスクのスクリプトを実行する前に、バックエンドのエンジンはロード（'nashorn:Mozilla_compat.js'）を呼び出します。

- **静的メソッドへのアクセス**

静的メソッドへのアクセスの柔軟性は Nashorn エンジンでは低下します。Rhino のバージョンのエンジンでは、静的メソッドにはクラス名（Java と同じ構文）からだけでなく、そのクラスのどのインスタンスからも（Java とは異なり）アクセスできました。

Rhino での静的メソッドへのアクセス

```
var myRBUtil = new com.cloupia.service.cIM.inframgr.i18n.RBUtil();
myRBUtil.getString();// No error
com.cloupia.service.cIM.inframgr.i18n.RBUtil.getString();// No error
```

Nashorn での静的メソッドへのアクセス

```
var myRBUtil = new com.cloupia.service.cIM.inframgr.i18n.RBUtil();
myRBUtil.getString();// No error
com.cloupia.service.cIM.inframgr.i18n.RBUtil.getString();// No error
```

- **ネイティブ JSON オブジェクトと com.cloupia.lib.util.JSON との比較**

  Nashorn 環境はネイティブ JSON オブジェクトで構成され、オブジェクトを JSON 形式に変換したり、またその逆に変換するための組み込み関数があります。Cisco UCS Director には、`com.cloupia.lib.util.JSON` と呼ばれる独自のバージョンの JSON オブジェクトがあります。

  Cisco UCS Director クラスをインポートすると、ネイティブ JSON オブジェクトへのアクセスは同じオブジェクト名が使用されているため、失われます。UCS Director とネイティブ JSON オブジェクト両方の使用をイネーブルにするために、UCS Director は名前 NativeJSON を使用してネイティブ クラスを保存します。たとえば、以下はネイティブ オブジェクトの静的メソッド コールです。

  ```
  NativeJSON.stringify(object myObj); OR
  NativeJSON.parse(String mystr);
  ```

- **文字列への新しい演算子の使用**

  オブジェクトの作成時に明示的にキーワード「new」を追加します。

  次に例を示します。

  ```
  var customName = new java.lang.String(input.name);
  var ai = new CMDB.AdditionalInfo();// static class
  ```

第 3 章

CloupiaScript インタープリタ

CloupiaScript インタープリタについて

CloupiaScript インタープリタは、ライブラリと API が組み込まれた JavaScript インタープリタです。CloupiaScript インタープリタを使用して、ワークフロー タスクを作成および実行することなく、CloupiaScript コードをテストすることができます。

CloupiaScript インタープリタには、次の組み込み関数が用意されています。

- `PrintObj()`：オブジェクトを引数として取り、オブジェクト内のすべてのプロパティとメソッドを出力します。出力された結果には、オブジェクト内の変数の名前と値、およびオブジェクトのすべての関数の名前が示されます。その後、いずれかのメソッド名で `toString()` を呼び出し、メソッド シグネチャを調べることができます。
- `Upload()`：ファイル名を引数として取り、ファイルの内容を CloupiaScript インタープリタにアップロードします。

CloupiaScript インタープリタの開始

CloupiaScript インタープリタを開くには、次の手順に従います。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。
ステップ 2 [カスタムワークフロータスク（Custom Workflow Tasks）] タブをクリックします。
ステップ 3 [インタープリタの起動（Launch Interpreter）] をクリックします。
[Cloupiaスクリプトインタープリタ（Cloupia Script Interpreter）] ダイアログボックスが表示されます。
ステップ 4 インタープリタ ダイアログの下部のテキスト入力フィールドをクリックします。
ステップ 5 JavaScript コードの行を入力して、Enter を押します。
コードが実行され、結果が表示されます。コードに構文エラーがある場合、そのエラーが表示されます。

コンテキストによる CloupiaScript インタープリタの開始

特定のカスタム タスクのコンテキストで JavaScript を評価できます。これを行うには、カスタム タスクを選択し、そのカスタム タスクを実行するために定義されたすべてのコンテキスト変数を使用して CloupiaScript インタープリタを起動します。

インタープリタを起動すると、インタープリタはユーザにカスタム タスクの入力フィールドに対する値を要求し、タスクの入力オブジェクトを入力します。実際にカスタム タスクを実行した場合に利用可能であったすべての変数が使用可能になります。

使用可能なコンテキストで CloupiaScript インタープリタを開くには、次の手順を実行します。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。
ステップ 2 [カスタムワークフロータスク（Custom Workflow Tasks）] タブをクリックします。
ステップ 3 JavaScript のテストの対象となるカスタム タスクを選択します。
ステップ 4 [コンテキストを使用してインタープリタを起動（Launch Interpreter with Context）] アクションをクリックします。
[インタープリタの起動（Launch Interpreter）] ダイアログボックスが表示されます。ここには、カスタム タスクの入力値を収集するための入力フィールドが示されます。入力フィールドは、選択したカスタム タスクに対して定義されたものです。
ステップ 5 フォームに入力値を入力します。
ステップ 6 [送信（Submit）] をクリックします。
ステップ 7 [送信（Submit）] をクリックします。

[Cloupiaスクリプトインタープリタ（Cloupia Script Interpreter）] ダイアログボックスが表示されます。

ステップ 8 インタープリタ ダイアログの下部のテキスト入力フィールドをクリックします。

ステップ 9 JavaScript コードの行を入力して、Enter を押します。
コードが実行され、結果が表示されます。コードに構文エラーがある場合、そのエラーが表示されます。

例：CloupiaScript インタープリタの使用

printObj 関数は、その関数に含まれているすべてのプロパティとメソッドを出力します。`functiontoString()` を呼び出して、関数に関する詳細を確認できます。次に、ReportContext クラスを調べて `ReportContext.setCloudName()` に関する詳細を取得する方法を示します。

```
session started
> importPackage(com.cloupia.model.cIM);
> var ctx = new ReportContext();
> printObj(ctx);
properties =
cloudName:null
class:class com.cloupia.model.cIM.ReportContext
filterId:null
id:null
targetCuicId:null
type:0
ids:[Ljava.lang.String;@4de27bc5
methods =
setIds
jdoReplaceField
jdoReplaceFields
toString
getCloudName
wait
getClass
jdoReplaceFlags
hashCode
jdoNewInstance
jdoReplaceStateManager
jdoIsDetached
notify
jdoGetVersion
jdoProvideField
jdoCopyFields
jdoGetObjectId
jdoGetPersistenceManager
jdoCopyKeyFieldsToObjectId
jdoGetTransactionalObjectId
getType
getFilterId
setType
jdoIsPersistent
equals
setCloudName
jdoNewObjectIdInstance
jdoIsDeleted
getTargetCuicId
setId
setFilterId
jdoProvideFields
jdoMakeDirty
jdoIsNew
requiresCloudName
```

```
getIds
notifyAll
jdoIsTransactional
getId
jdoReplaceDetachedState
jdoIsDirty
setTargetCuicId
jdoCopyKeyFieldsFromObjectId

> var func = ctx.setCloudName;
> func
void setCloudName(java.lang.String)
> func.toString();
function setCloudName() {/*
void setCloudName(java.lang.String)
*/}
```

第 4 章

カスタム ワークフロー タスクの作成

カスタム ワークフロー入力について

Cisco UCS Director Orchestrator は、カスタム タスクのために詳細に定義された入力タイプのリストを提供します。入力タイプのリストを使用してカスタム ワークフロータスク用の入力を定義できます。Cisco UCS Director では、カスタム ワークフロー タスク用にカスタマイズされたワークフロー入力を作成することもできます。既存の入力タイプを複製して変更することで、新しい入力タイプを作成できます。

前提条件

カスタム タスクを記述する前に、次の前提条件を満たす必要があります。

- Cisco UCS Director がインストール済みで、システム上で動作していること。Cisco UCS Director をインストールする方法の詳細については、『Cisco UCS Director Installation and Configuration Guide』を参照してください。
- 管理者権限でログインしていること。カスタム タスクを作成および変更するときはこのログインを使用する必要があります。

カスタム ワークフローの入力値の作成

カスタム ワークフロー タスクのカスタム入力を作成できます。作成した入力は、カスタム ワークフロータスクの作成時にカスタムタスク入力にマッピングできる、入力タイプのリストに表示されます。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [カスタムワークフローの入力値（Custom Workflow Inputs）] タブを選択します。

ステップ 3 [追加（Add）] アイコンをクリックします。

ステップ 4 [カスタム ワークフローの入力値の追加（Add Custom Workflow Input）] ダイアログボックスで、次のフィールドに入力します。

名前	説明
[カスタム入力タイプ名（Custom Input Type Name）] フィールド	カスタム入力タイプの固有名。
[入力タイプ（Input Type）] ボタン	入力のタイプを選択します。選択した入力に基づいて、他のフィールドが表示されます。たとえば、入力タイプとして [電子メールアドレス（Email Address）] を選択すると、値のリスト（LOV）が表示されます。カスタム入力の値を制限するには、新規フィールドを使用します。

ステップ 5 [送信（Submit）] をクリックします。
カスタム ワークフロー入力が Cisco UCS Director に追加され、入力タイプのリストで使用できるようになります。

カスタム ワークフロー入力のクローニング

Cisco UCS Director 内の既存のカスタム ワークフロー入力を使用してカスタム ワークフロー入力を作成できます。

はじめる前に

カスタム ワークフロー入力が Cisco UCS Director で利用可能である必要があります。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [カスタムワークフローの入力値（Custom Workflow Inputs）] タブを選択します。

ステップ 3 クローンを作成する必要があるカスタム ワークフロー入力を選択します。
[カスタムワークフローの入力値（Custom Workflow Inputs）] テーブルの上部に [複製（Clone）] アイコンが表示されます。

ステップ 4 [複製（Clone）] アイコンをクリックします。

ステップ 5 [カスタム入力タイプ名（Custom Input Type Name）] フィールドに新しい入力の名前を入力します。

ステップ 6 [カスタムワークフロー入力の複製（Clone Custom Workflow Input）] ダイアログボックスの他のコントロールを使用して、新しい入力をカスタマイズします。

ステップ 7 [送信（Submit）] をクリックします。
カスタム ワークフロー タスク入力は確認後にクローン作成され、カスタム ワークフロー タスクで使用できるようになります。

カスタム タスクの作成

カスタム タスクを作成するには、次の手順を実行します。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [カスタムワークフロータスク（Custom Workflow Tasks）] タブを選択します。

ステップ 3 [追加（Add）] アイコンをクリックします。

ステップ 4 [カスタムワークフロータスクの追加（Add Custom Workflow Task）] ダイアログボックスで、次のフィールドに入力します。

名前	説明
[タスク名（Task Name）] フィールド	カスタム ワークフロー タスクの固有名。
[タスクラベル（Task Label）] フィールド	カスタム ワークフロー タスクを識別するラベル。

名前	説明
[タスクのアクティブ化（Activate Task）] チェックボックス	オンにした場合、カスタム ワークフロー タスクは Orchestrator に登録され、ワークフローで直ちに使用できるようになります。
[カテゴリへの登録（Register Under Category）] フィールド	カスタム ワークフロー タスクが登録されているカテゴリ。
[簡潔な説明（Brief Description）] フィールド	カスタム ワークフロー タスクの説明。
[詳細な説明（Detailed Description）] フィールド	カスタム ワークフロー タスクの詳細な説明。

ステップ 5 [次へ（Next）] をクリックします。
[カスタムワークフロータスク入力（Custom workflow Tasks Inputs）] ウィンドウが表示されます。

ステップ 6 [追加（Add）] アイコンをクリックします。

ステップ 7 [入力フィールドへのエントリの追加（Add Entry to Inputs）] ダイアログボックスで、次のフィールドに入力します。

名前	説明
[入力フィールド名（Input Field Name）] フィールド	フィールドの固有名。名前の先頭は英字にする必要があります。名前に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（Input Field Label）] フィールド	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（Input Field Type）] ドロップダウン リスト	入力パラメータのデータ型を選択します。
[入力タイプにマッピング（Map to Input Type）] ボタン	別のタスク出力またはグローバル ワークフロー入力からマッピングできる入力タイプを選択します。
[必須（Mandatory）] チェックボックス	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。
[入力フィールドのサイズ（Input Field Size）] ドロップダウン リスト	テキストおよび表形式の入力のフィールド サイズを選択します。
[入力フィールドヘルプ（Input Field Help）] フィールド	（オプション）フィールド上にマウスのポインタを置いた場合に表示される説明。

名前	説明
[入力フィールド注記（Input Field Annotation）] フィールド	（オプション）入力フィールドに対するヒント テキスト。
[フィールドグループ名（Field Group Name）] フィールド	指定した場合は、グループ名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。
[複数の入力（Multiple Input）] チェックボックス	オンにした場合、入力フィールドは入力フィールド タイプに基づいて複数の値を受け入れます。 • LOV の場合：入力フィールドは、複数の入力値を受け入れます。 • テキスト フィールドの場合：入力フィールドは複数行のテキスト フィールドになります。
[テキストフィールド属性（Text Field Attributes）] エリア：入力フィールド タイプがテキスト フィールドの場合、次のフィールドに値を入力します。	
[入力値の最大長（Maximum Length of Input）] フィールド	入力フィールドに入力できる文字の最大数を指定します。
[LOV属性（LOV Attributes）] エリア：入力タイプが値のリスト（LOV）またはオプション ボタン付きの LOV の場合、次のフィールドに値を入力します。	
[値のリスト（List of Values）] フィールド	組み込み LOV の値のカンマ区切りリスト。
[LOVのプロバイダー名（LOV Provider Name）] フィールド	非組み込み LOV の LOV プロバイダーの名前。
[テーブル属性（Table Attributes）] エリア：入力フィールド タイプが [テーブル（Table）]、[ポップアップテーブル（Popup Table）]、または [選択チェックボックス付きのテーブル（Table with selection checkbox）] の場合、次のフィールドに値を入力します。	
[テーブル名（Table Name）] フィールド	テーブル フィールド タイプに対する表形式レポートの名前。
[フィールド入力の検証（Field Input Validation）] エリア：選択したデータ型に応じて、次の 1 つ以上のフィールドが表示されます。フィールドに値を入力して、入力フィールドの検証方法を指定します。	
[入力バリデータ（Input Validator）] ドロップダウン リスト	ユーザ入力のバリデータを選択します。
[正規表現（Regular Expression）] フィールド	入力値を照合する正規表現パターン。

名前	説明
[正規表現メッセージ（Regular Expression Message）] フィールド	正規表現の検証が失敗したときに表示されるメッセージ。
[最小値（Minimum Value）] フィールド	最小の数値。
[最大値（Maximum Value）] フィールド	最大の数値。

ステップ 8 [送信（Submit）] をクリックします。
エントリが正常に追加されたことを示すメッセージが表示されます。

ステップ 9 [OK] をクリックします。

ステップ 10 入力にさらにエントリを追加するには [追加（Add）] アイコンをクリックします。

ステップ 11 [次へ（Next）] をクリックします。
[カスタムワークフロータスク出力（Custom Workflow Tasks Outputs）] ウィンドウが表示されます。

ステップ 12 [追加（Add）] アイコンをクリックします。

ステップ 13 [出力にエントリを追加（Add Entry to Outputs）] ダイアログボックスで、次のフィールドに入力します。

名前	説明
[出力フィールド名（Output Field Name）] フィールド	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（Output Field Description）] フィールド	出力フィールドの説明。
[出力フィールドのタイプ（Output Field Type）] ボタン	出力のタイプを選択します。この値によって、他のタスク入力への出力のマッピング方法が決まります。

ステップ 14 [送信（Submit）] をクリックします。
エントリが正常に追加されたことを示すメッセージが表示されます。

ステップ 15 [OK] をクリックします。

ステップ 16 出力にさらにエントリを追加するには [追加（Add）] アイコンをクリックします。

ステップ 17 [次へ（Next）] をクリックします。
[コントローラ（Controller）] ウィンドウが表示されます。

ステップ 18 （オプション）[追加（Add）] アイコンをクリックしてコントローラを追加します。

ステップ 19 [コントローラにエントリを追加（Add Entry to Controller）] ダイアログボックスで、次のフィールドに入力します。

名前	説明
[メソッド（Method）] ドロップダウン リスト	marshalling または unmarshalling メソッドを選択して、カスタム ワークフロー タスクに対する入力と出力のいずれかまたは両方をカスタマイズします。メソッドは次のいずれかです。 • [beforeMarshall]：このメソッドを使用して、入力フィールドを追加または設定し、ページ（フォーム）上に LOV を動的に作成および設定します。 • [afterMarshall]：このメソッドを使用して入力フィールドを表示または非表示にします。 • [beforeUnmarshall]：このメソッドを使用して入力値の形式を変換します。たとえば、データベースに送信する前にパスワードを暗号化する場合などです。 • [afterUnmarshall]：このメソッドを使用して、ユーザ入力を検証し、ページ上にエラー メッセージを設定します。
[スクリプト（Script）] テキスト エリア	[メソッド（Method）] ドロップダウン リストから選択したメソッドに対して、ここに GUI カスタマイゼーション スクリプトのコードを追加します。 （注） さらにメソッドにコードを追加するには [追加（Add）] アイコンをクリックします。

ステップ 20 [送信（Submit）] をクリックします。
エントリが正常に追加されたことを示すメッセージが表示されます。

ステップ 21 [次へ（Next）] をクリックします。
[スクリプト（Script）] ウィンドウが表示されます。

ステップ 22 [実行言語（Execution Language）] ドロップダウン リストから言語を選択します。

ステップ 23 [スクリプト（Script）] フィールドに、カスタム ワークフロー タスクの CloupiaScript コードを入力します。

ステップ 24 [スクリプトの保存（Save Script）] をクリックします。

ステップ 25 [送信（Submit）] をクリックします。
カスタム ワークフロー タスクが作成され、ワークフローで使用できるようになります。

ワークフロー、カスタム タスク、スクリプト モジュール、およびアクティビティのインポート

Cisco UCS Director にアーティファクトをインポートするには、次の手順を実行します。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [オーケストレーション（Orchestration）] ペインで [ワークフロー（Workflows）] タブをクリックします。

ステップ 3 [インポート（Import）] アクションをクリックします。

ステップ 4 [インポート（Import）] ダイアログボックスで、[アップロード（Upload）] をクリックします。

ステップ 5 [ファイルのアップロード（File Upload）] ダイアログで、[コンピュータからファイルをクリックして選択する（Click and select a file from your computer）] をクリックします。

ステップ 6 インポート ファイルを選択します。Cisco UCS Director のインポートおよびエクスポート ファイルには `.wfdx` というファイル拡張子が付いています。
ファイルがアップロードされると、[ファイルのアップロード（File Upload）] ダイアログに [ファイルが使用できるようになりました（File ready for use）] と表示されます。

ステップ 7 [ファイルのアップロード（File Upload）] ダイアログを閉じます。

ステップ 8 [次へ（Next）] をクリックします。
[インポート（Import）] ダイアログに、アップロードされたファイルに含まれている Cisco UCS Director オブジェクトのリストが表示されます。

ステップ 9 （任意）オブジェクトがワークフローフォルダにすでに存在する名前と重複する場合のオブジェクトの処理方法を指定します。[インポート（Import）] 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。

名前	説明
[ワークフロー（Workflows）] ドロップダウン リスト	次のオプションの中から、同じ名前が付けられたワークフローの処理方法を指定します。 • [置換（Replace）] : 既存のワークフローをインポートされたワークフローに置き換えます。 • [両方保持（Keep Both）] : 新しいバージョンとしてワークフローをインポートします。 • [省略（Skip）] : ワークフローをインポートしません。
[カスタム タスク（Custom Tasks）] ドロップダウン リスト	次のオプションの中から、同じ名前が付けられたカスタム タスクの処理方法を指定します。 • 置換（Replace） • 両方保持（Keep Both） • 省略（Skip）

名前	説明
[スクリプト モジュール（Script Modules）] ドロップダウン リスト	次のオプションの中から、同じ名前が付けられたスクリプト モジュールの処理方法を指定します。 • 置換（Replace） • 両方保持（Keep Both） • 省略（Skip）
[アクティビティ（Activities）] ドロップダウンリスト	次のオプションの中から、同じ名前が付けられたアクティビティの処理方法を指定します。 • 置換（Replace） • 両方保持（Keep Both） • 省略（Skip）
[ワークフローをフォルダにインポート（Import Workflows to Folder）] チェックボックス	ワークフローをインポートするには、このチェックボックスをオンにします。このチェックボックスをオンにせず、ワークフローの既存バージョンがない場合、そのワークフローはインポートされません。
[フォルダの選択（Select Folder）] ドロップダウン リスト	ワークフローをインポートするフォルダを選択します。ドロップダウンリストから [新しいフォルダ...（New Folder..）] を選択した場合は、[新しいフォルダ（New Folder）] フィールドが表示されます。
[新しいフォルダ（New Folder）] フィールド	インポート フォルダとして作成する新しいフォルダの名前を入力します。

ステップ 10 [インポート（Import）] をクリックします。

ワークフロー、カスタム タスク、スクリプト モジュール、およびアクティビティのエクスポート

Cisco UCS Director からアーティファクトをエクスポートするには、次の手順を実行します。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。
ステップ 2 [オーケストレーション（Orchestration）] ペインで [ワークフロー（Workflows）] タブをクリックします。
ステップ 3 [ワークフロー（Workflows）] タブで、[エクスポート（Export）] をクリックします。
ステップ 4 [ワークフローの選択（Select Workflows）] 画面で、エクスポートするワークフローを選択します。
ステップ 5 [次へ（Next）] をクリックします。
ステップ 6 [カスタムタスクの選択（Select Custom Tasks）] 画面で、エクスポートするカスタム タスクを選択します。
ステップ 7 [次へ（Next）] をクリックします。
ステップ 8 [エクスポート：スクリプトモジュールの選択（Export: Select Script Modules）] 画面で、エクスポートするスクリプト モジュールを選択します。
ステップ 9 [次へ（Next）] をクリックします。
ステップ 10 [エクスポート：アクティビティの選択（Export: Select Activities）] 画面で、エクスポートするアクティビティを選択します。
ステップ 11 [エクスポート：確認（Export: Confirmation）] 画面で、次のフィールドに値を入力します。

名前	説明
[エクスポート実行ユーザ（Exported By）] テキスト フィールド	エクスポートを担当する人物の名前またはメモ。
[コメント（Comments）] テキスト エリア	このエクスポートに関するコメント。
[エクスポートされたファイルの名前（Exported File Name）] テキスト フィールド	ローカル システム上のファイルの名前。ベースのファイル名のみを入力します。ファイル タイプの拡張子（`.wfdx`）は自動的に付けられます。

ステップ 12 [エクスポート（Export）] をクリックします。

ファイルを保存するよう求められます。

タスク ライブラリからのカスタム ワークフロー タスクのクローニング

タスク ライブラリ内のタスクを複製してカスタム タスクの作成に使用できます。

複製されたタスクは元のタスクと同じタスクの入出力を備えたフレームワークです。ただし、複製したタスクはフレームワークだけであることに注意してください。これは、CloupiaScript で新しいタスクのすべての機能を記述する必要があることを意味します。

また、ドロップダウン リストや値リストなどのリスト入力の選択値は、リストの値がシステムに依存していない場合に限り、複製したタスクに引き継がれることにも注意してください。既存システムの名前やIP アドレスなどはシステムに依存しており、Cisco UCS Director でサポートされる構成オプションなどは依存していません。たとえば、ユーザ グループ、クラウド名、ポート グループは、システムに依存しており、ユーザ ロール、クラウド タイプ、ポート グループ タイプは依存していません。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [カスタムワークフロータスク（Custom Workflow Tasks）] タブを選択します。

ステップ 3 [タスクライブラリよりクローン（Clone From Task Library）] をクリックします。

ステップ 4 [タスクライブラリよりクローン（Clone From Task Library）] ダイアログボックスで、[選択（Select）] をクリックします。

ステップ 5 タスク リストからタスクを選択します。

ステップ 6 [選択（Select）] をクリックします。
タスク ライブラリからカスタム ワークフロー タスクが作成されます。新しいカスタム タスクはカスタム ワークフロー タスク レポートの最後のカスタム タスクです。新しいカスタム タスクの名前は、複製されたタスクにちなんで付けられ、日付が追加されます。

次の作業

カスタム ワークフロー タスクを編集します。

カスタム ワークフロー タスクのクローニング

Cisco UCS Director 内の既存のカスタム ワークフロー タスクを使用してカスタム ワークフロー タスクを作成できます。

はじめる前に

カスタム ワークフロー タスクが Cisco UCS Director で利用可能である必要があります。

ステップ 1 メニュー バーで、[ポリシー（Policies）] > [オーケストレーション（Orchestration）] を選択します。

ステップ 2 [カスタムワークフロータスク（Custom Workflow Tasks）] タブを選択します。

ステップ 3 複製するカスタム ワークフロー タスクを選択します。
[カスタムワークフロータスク（Custom Workflow Tasks）] テーブルの上部に [複製（Clone）] アイコンが表示されます。

ステップ 4 [複製（Clone）] アイコンをクリックします。

ステップ 5 [カスタムワークフロータスクの複製（Clone Custom Workflow Task）] ダイアログボックスで、必須フィールドを更新します。

ステップ 6 [次へ（Next）] をクリックします。
カスタム ワークフロー タスク用に定義された入力が表示されます。

ステップ 7 タスク入力を編集します。

ステップ 8 [次へ（Next）] をクリックします。
タスク出力を編集します。

ステップ 9 新規出力エントリを追加するには [追加（Add）] アイコンをクリックします。

ステップ 10 [次へ（Next）] をクリックします。

ステップ 11 コントローラ スクリプトを編集します。「カスタム ワークフロー タスク入力の制御」を参照してください。

ステップ 12 [次へ（Next）] をクリックします。

ステップ 13 カスタム タスクをカスタマイズするには、タスクのスクリプトを編集します。

ステップ 14 [送信（Submit）] をクリックします。

カスタム ワークフロー タスク入力の制御

コントローラの使用

Cisco UCS Director のコントローラ インターフェイスを使用して、カスタム タスク入力の外観と動作を変更できます。

コントローラの使用シナリオ

以下の場合にコントローラを使用します。

- 値のリストの詳細な制御、表形式の値のリスト、およびその他の入力コントロールなど、ユーザに表示される GUI の複雑な表示/非表示動作を実装する場合。
- 複雑なユーザ入力検証ロジックを実装する場合。

入力コントローラを使用すると、次の操作を実行できます。

- GUI 制御の表示または非表示：条件に基づいて、チェック ボックス、テキスト ボックス、ドロップダウン リスト、ボタンなどのさまざまな GUI フィールドの表示/非表示を動的に切り替えることができます。たとえば、ユーザがドロップダウン リストから [UCSM] を選択したときに Cisco UCS Manager に対するユーザ クレデンシャルの入力を促すプロンプトが出されるようにしたり、サーバで使用可能なポートのみがドロップダウン リストの値のリスト（LOV）に表示されるようにしたりできます。
- フォーム フィールド検証：[ワークフローデザイナ（Workflow Designer）] でワークフローの作成時または編集時に、ユーザが入力したデータを検証できます。ユーザが無効な値を入力した場合は、エラーを表示できます。ユーザの入力データに変更を加えてからデータベースまたはデバイスに保存できます。
- 値のリストの動的取得：Cisco UCS Director オブジェクトから値のリストを動的に取得して、GUI フォーム オブジェクトに入力できます。

GUI フォーム オブジェクトの整列化および非整列化

コントローラは、ワークフロー デザイナーのタスク入力インターフェイスにあるフォームに常に関連付けられています。フォームとコントローラの関連付けは 1 対 1 です。コントローラは、整列化と非整列化という 2 つのステージで機能します。どちらのステージにも、前後にサブステージがあります。コントローラを使用するには、コントローラのスクリプトを使用して、関連する GUI フォーム オブジェクトの整列化（UI フォーム フィールドの制御）や非整列化（ユーザ入力の検証）を実行します。

次の表は、各ステージの内容を説明しています。

ステージ	サブステージ
[整列化（Marshalling）]：フォーム フィールドの表示/非表示を切り替えるために使用します。また、LOV および表形式の LOV を詳細に制御する場合にも使用します。	[beforeMarshall]：入力フィールドを追加または設定して、ページ（フォーム）上に動的に LOV を作成および設定するために使用します。 [afterMarshall]：入力フィールドの表示/非表示を切り替えるために使用します。
[非整列化（Unmarshalling）]：ユーザ入力の検証に使用します。	[beforeUnmarshall]：入力値の形式を変換するために使用します。たとえば、データベースに送信する前にパスワードを暗号化する場合などです。 [afterUnmarshall]：ユーザ入力を検証し、ページ上に表示するエラー メッセージを設定するために使用します。

コントローラ スクリプトの構築

コントローラを使用するために追加のパッケージをインポートする必要はありません。

コントローラのメソッドにはパラメータを渡しません。代わりに、Cisco UCS Director フレームワークでは、整列化および非整列化に次のパラメータを使用できるようになっています。

パラメータ	説明	例
Page	すべてのタスク入力を格納するページまたはフォーム。このパラメータを使用することで、次の作業が実行できます。 • GUI フォームで入力値を取得または設定します。 • GUI フォームで入力を表示または非表示にします。	`page.setHidden(id + ".portList", true);` `page.setValue(id + ".status", "No Port is up.  Port List is Hidden");`
id	フォーム入力フィールドの固有識別子。フレームワークにより生成され、フォーム入力フィールド名と共に使用できます。	`page.setValue(id + ".status", "No Port is up.  Port List is Hidden");// here 'status' is the name of the input field.`
Pojo	POJO（Plain Old Java Object）は、入力フォームを表す Java Bean です。すべての GUI ページには対応する POJO が必要で、各 POJO がそのフォームからの値を保持します。POJO を使用して、値をデータベースに保存したり外部デバイスに送信したりします。	`pojo.setLunSize(asciiValue); //set the value of the input field 'lunSize'`

コントローラの機能を使用する、運用可能なコード サンプルについては、例：コントローラの使用, （26 ページ） を参照してください。

例：コントローラの使用

次のコード例は、beforeMarshall、afterMarshall、beforeUnmarshall および afterUnmarshall などの各種メソッドを使用して、カスタム ワークフロー タスク内でコントローラの機能を実装する方法を示しています。

```
/*
Method Descriptions:

Before Marshall: Use this method to add or set an input field and dynamically create and
set the LOV on a page(form).
After Marshall: Use this method to hide or unhide an input field.
Before UnMarshall: Use this method to convert an input value from one form to another form,
 for example, when you want to encrypt the password before sending it to the database.
After UnMarshall: Use this method to validate a user input and set the error message on the
 page.
```

```
*/

//Before Marshall:

/*
Use the beforeMarshall method when there is a change in the input field or to dynamically
create LOVs and to set the new input field on the form before it gets loaded.
 In the example below, a new input field 'portList' is added on the page before the form
is displayed in a browser.
*/
importPackage(com.cloupia.model.cIM);
importPackage(java.util);
importPackage(java.lang);

var portList = new ArrayList();
var lovLabel = "eth0";
var lovValue = "eth0";

var portListLOV = new Array();
portListLOV[0] = new FormLOVPair(lovLabel, lovValue);//create the lov input field
//the parameter 'page' is used to set the input field on the form
page.setEmbeddedLOVs(id + ".portList",  portListLOV);// set the input field on the form
```

```
//After Marshall :
/*
Use this method to hide or unhide an input field.
*/
page.setHidden(id + ".portList",  true); //hide the input field 'portList'.
page.setValue(id + ".status", "No Port is up.  Port List is Hidden");
page.setEditable(id + ".status", false);
```

```
//Before Unmarshall :

/*
  Use the beforeUnMarshall method to read the user input and convert it to another form
before inserting into the database. For example, you can read the password and store the
password in the database after converting it into base64 encoding, or read the employee
name and convert to the employee Id when the employee name is sent to the database.

In the code example below the lun size is read and converted into an ASCII value.

*/
importPackage(org.apache.log4j);
importPackage(java.lang);
importPackage(java.util);

var size = page.getValue(id + ".lunSize");
var logger = Logger.getLogger("my logger");

if(size != null){
                logger.info("Size value "+size);
                if((new java.lang.String(size)).matches("\\d+")){
                                var byteValue = size.getBytes("US-ASCII"); //convert the
lun size and get the  ASCII character array
                                var asciiValueBuilder = new StringBuilder();
                                for (var i = 0; i < byteValue.length; i++) {
                                                asciiValueBuilder.append(byteValue[i]);
                                }
                                var asciiValue = asciiValueBuilder.toString()+" - Ascii
value"
                                //id + ".lunSize" is the identifier of the input field
                                page.setValue(id + ".lunSize",asciiValue); //the parameter
 'page' is used to set the value on the input field .
                                pojo.setLunSize(asciiValue); //set the value on the pojo.
This pojo will be send to DB or external device.
                }
}
```

```
// After unMarshall :

/*
Use this method to validate and set an error message.
*/
importPackage(org.apache.log4j);
importPackage(java.lang);
importPackage(java.util);

//var size = pojo.getLunSize();
var size = page.getValue(id + ".lunSize");
var logger = Logger.getLogger("my logger");
logger.info("Size value "+size);
if (size > 50) { //validate the size
                page.setError(id+".lunSize", "LUN Size can not be more than 50MB "); //set
 the error message on the page
                page.setPageMessage("LUN Size can not be more than 50MB");
                //page.setPageStatus(2);
}
```

例：カスタム タスクの作成と実行

カスタム タスクを作成するには、次の手順を実行します。

ステップ 1 [ポリシー（Policies）]>[オーケストレーション（Orchestration）]>[カスタムワークフロータスク（Custom Workflow Tasks）] に移動します。

ステップ 2 [追加（Add）] をクリックして、カスタム タスク情報を入力します。

ステップ 3 [次へ（Next）] をクリックします。
[Cloupiaスクリプトインタープリタ（Cloupia Script Interpreter）] ダイアログボックスが表示されます。

ステップ 4 [+] をクリックして入力の詳細を追加します。

ステップ 5 [送信（Submit）] をクリックします。

ステップ 6 [次へ（Next）] をクリックします。
カスタム タスクの出力ウィンドウが表示されます。

ステップ 7 [次へ（Next）] をクリックします。
カスタム タスクのコントローラ ウィンドウが表示されます。

ステップ 8 [次へ（Next）] をクリックします。
スクリプト ウィンドウが表示されます。

ステップ 9 実行言語として [JavaScript] を選択し、次のスクリプトを入力して実行します。

```
logger.addInfo("Hello World!");
logger.addInfo("Message "+input.message);
```

message は入力フィールドの名前です。

ステップ 10 [送信（Submit）] をクリックします。

カスタム タスクが定義され、カスタム タスク リストに追加されます。

ステップ 11 [ワークフロー（Workflows）] タブに移動します。

ステップ 12 [追加（Add）] をクリックしてワークフローを追加します。

ステップ 13 ワークフローが作成されたら、「Hello world カスタム タスク」をワークフロー デザイナにドラッグ アンド ドロップします。

ステップ 14 「Hello World カスタム タスク」をデザイナに追加します。

ステップ 15 [ワークフローの検証（Validate workflow）] をクリックします。

ステップ 16 [今すぐ実行（Execute Now）] をクリックし、[送信（Submit）] をクリックします。

ステップ 17 [サービスリクエスト（Service Request）] ログ ウィンドウでログ メッセージを確認します。

第 5 章

レポートの管理

レポートへのアクセス

CloupiaScript を使用してレポートにアクセスできます。レポート データを使用することで、後続タスクについて動的に意思決定できます。

たとえば、関連付けなしの 32GB を超える Cisco UCS B シリーズ ブレード サーバを割り当てるには、次のスクリプトを使用して特定の Cisco UCS Manager で管理されているすべての Cisco UCS サーバのリストを問い合わせます。このスクリプトは値のサブセットを抜粋してフィルタする方法の例です。`getReportView(reportContext, reportName)` 関数は、reportContext および reportName を引数として取得し、表形式でコンテンツを表示する TableView オブジェクトを返します。

```
importPackage(java.lang);
importPackage(java.util);
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);
}

// following are only sample values and need to be modified based on actual UCSM account
name
var ucsmAccountName = "ucs-account-1";

// report name is obtained from Report Meta. No need to change unless you need to access a
 different report
var reportName = "UcsController.allservers.table_config";

var repContext = util.createContext("ucsm", null, ucsmAccountName);
// Enable Developer Menu in UCSD and find reportName in the Report Metadata for the specific
 report
// Creating a ReportContext
// @param contextName
// Refer to UCSD API Guide for the available contexts
// @param cloud
// should be null unless contextName is "cloud" or "host node"

// @param value
// identifier of the object that is going to be referenced

Report var report = getReportView(repContext, reportName);

// Get only the rows for which Server Type column value is B-Series
report = report.filterRowsByColumn("Server Type", "B-Series", false);

// now look for unassociated servers only
report = report.filterRowsByColumn("Operation State", "unassociated", false);

// Make sure servers are actually in available state
report = report.filterRowsByColumn("Availability", "available", false);

var matchingIds = [];
var count = 0;


// Now look for Servers with memory of 32 GB or more
for (var i=0; i<report.rowCount(); i++)
{

      var memory = Integer.parseInt(report.getColumnValue(i, "Total Memory (MB)"));

      logger.addDebug("Possible Server "+report.getColumnValue(i, "ID")+", mem="+memory);

      if (memory >= 32*1024)
      {
               matchingIds[count++] = report.getColumnValue(i, "ID");
      }
}

if (count == 0)
{
     ctxt.setFailed("No servers matched the criteria");
     ctxt.exit();
}

// Now randomly pick one of the item from the filtered list
var id = matchingIds[Math.round(Math.random()*count)];
logger.addInfo("Allocated server "+id);

// Save the Server-ID to the global inputs
ctxt.updateInput("SELECTED_UCS_SERVER_ID", id);
```

表形式のレポートへのアクセス

`getTabularReport (reportName, reportContext)` API を使用して表形式のレポートにアクセスする場合は、次のいずれかの方法でユーザ インターフェイス（UI）にレポートの詳細を表示できます。

- [レポートのカスタマイズ（Reports Customization）] タブ：レポートのカスタマイズ タブにアクセスするには [管理（Administration）] > [ユーザインターフェイス設定（User Interface Settings）] を選択し、[レポートのカスタマイズ（Reports Customization）] を選択します。カスタマイズ タブには、メニュー、コンテキスト、レポート タイプなどのレポートの詳細が表示されます。テーブルの列をカスタマイズするには[テーブルの列のカスタマイズ（Customize Table Columns）] アイコンをクリックし、表示される列の項目のチェックボックスを選択します。たとえば、レポート ID を表示するには [ID] チェックボックスを選択します。
- レポート メタデータ：開発者メニューが有効の場合のみ、レポート メタデータのリンクが UI に表示されます。

レポートの詳細で重要なものを次に説明します。

- API レポート ID：userAPIGetTabularReport API を使用する場合、[API レポートID（API report ID）] 列を使用して、REST URL で使用される reportID パラメータの値を取得できます。この REST API は、Web ブラウザまたは他の REST クライアント アプリケーションから表形式のレポートを取得するために使用されます。
- ID：[ID] 列にはレポート名が表示されます。Cloupia スクリプトで getTabularReport API を使用する場合、reportName パラメータの取得に [ID] 列を使用できます。このパラメータは getConfigTableReport API でも使用されます。
- コンテキスト：ReportContext のコンストラクタには contextName および contextValue の 2 つの入力パラメータが必要です。標準のコンテキストには util.createContext("contextName", null, "instanceName") を使用します。たとえば、util.createContext("vm",null, vmId) であれば、vmId には UCS Director の VM を一意に識別する整数値の VM ID 値を指定します。クラウド コンテキストには util.createContext("contextName","cloudInstanceName", null) または util.createContext("contextName",null,"cloudInstanceName") を使用します。たとえば、util.createContext("cloud","All Clouds",null) または util.createContext("cloud", null, "All Clouds") です。

レポートのメール送信

Cloupia スクリプトを使用してユーザにレポートをメール送信できます。レポートを定期的にメール送信する必要がある場合は、ワークフローのワークフロー スケジュールを任意の頻度で設定できます。

ワークフロー入力変数 *Email Address* で指定されているユーザに対して、電源がオンになっているすべての VM のリストをメール送信するには、次のスクリプトを使用します。

```
importPackage(java.util);
importPackage(java.lang);
importPackage(java.io);
importPackage(com.cloupia.model.cEvent.notify);
```

```
importPackage(com.cloupia.model.cIM);
importPackage(com.cloupia.lib.util.mail);
importPackage(com.cloupia.fw.objstore);
importPackage(com.cloupia.lib.util.managedreports);

function getMailSettings()
{
      return ObjStoreHelper.getStore((new MailSettings()).getClass()).getSingleton();
}


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);
}

// Assume the To Email Addres is in the input variable 'Email Address'
var toEmail = [ ctxt.getInput("Email Address") ];

var message = new EmailMessageRequest();
message.setToAddrs(toEmail);
message.setSubject("VM List Reportl");
message.setFromAddress("no-reply@cisco.com");

var buffer = new StringWriter();
var printer = new PrintWriter(buffer);

// Formatter exists in multiple packages, so it needs fully qualified name
var formatter = new com.cloupia.lib.util.managedreports.Formatter(new File("."), printer);

var reportName  = "GLOBAL_VM_LIST_REPORT";
var repContext  = util.createContext("global", null, null);
var report = getReportView(repContext, reportName);

// Filter Active State VMs
report = report.filterRowsByColumn("Power State", "ON", false);
formatter.printTable(report);

printer.close();

var body = "<head><style type='text/css'>";

// Specify CSS for the report
```

```
body = body + "table { font-family: Verdana, Geneva, sans-serif; font-size: 12px; border:
thin solid #039; border-spacing: 0; background: #ffffff; } ";


body = body + " th { background-color: #6699FF; color: white; font-family: Verdana, Geneva,
 sans-serif; font-size: 10px; font-weight: bold; border-color: #CCF; border-style: solid;
border-width: 1px 1px 0 0; margin: 0; padding: 5px; } ";

body = body + " td { font-family: Verdana, Geneva, sans-serif; font-size: 10px; border-color:
 #CCF; border-style: solid; border-width: 1px 1px 0 0; margin: 0; padding: 5px; background:
 #ffffff; }";

body = body + "</style></head>";
body = body+ "<body><h1>List of Powered ON VMs</h1><br>" + buffer.toString();

message.setMessageBody(body);

logger.addInfo("Sending email");

// Now, send the report via email. First parameter is just a label used in the  internal
logs
MailManager.sendEmail("VM List Report", getMailSettings(), message);
```

第 6 章

ベスト プラクティス

ロールバック スクリプトの作成

カスタムタスクのスクリプトを作成するときは、対応するロールバック スクリプトを作成することを推奨します。ロールバック スクリプトは、カスタムタスクのスクリプトに行われた変更を元に戻します。たとえば、カスタムタスクがリソースを作成した場合、ロールバック スクリプトはそのリソースを削除します。

もちろん、多くのロールバック シナリオにはカスタムタスクが実行される前のシステムの状態に関する情報が必要です。CloupiaScript ライブラリには `ChangeTracker` API が含まれており、それを使用してカスタム タスクの影響を無効にできます。`ChangeTracker` API を使用して、リソースを作成する前に状態情報を収集する `UndoableResource` オブジェクトを作成します。ロールバック時に、`UndoableResource` はこの情報を使用して、リソースを前の状態に復元します。

`ChangeTracker` API には、リソースの変更および削除のロールバックをそれぞれ有効にするための 2 つのメソッドが含まれています。

- `ChangeTracker.undoableResourceModified()`
- `ChangeTracker.undoableResourceDeleted()`

ロールバック スクリプトを作成するために ChangeTracker API を使用する方法の例については、次の URL で入手可能な『*Cisco UCS DirectorCloupiaScript Cookbook*』を参照してください。
http://www.cisco.com/c/en/us/support/servers-unified-computing/ucs-director/products-programming-reference-guides-list.html