REST API の管理

この章は、次の項で構成されています。

REST API

Cisco UCS Director Open Automation ツールを使用して、モジュールとして独自の Cisco UCS Director 機能を開発した場合、Cisco UCS Director GUI でそのモジュールの REST API サポートを開発および公開できます。

以下は、Cisco UCS Director REST API を使用する際の用語です。

- MO：エンティティは管理対象オブジェクト（MO）としてインスタンス化されます。作成または更新処理は特定の MO を対象にする必要があります。MO は API を介して公開されます。
- クラス：管理情報ツリーのオブジェクトのプロパティおよび状態を定義するテンプレート。
- 属性：属性とは、同じクラス内のすべてのオブジェクトの特徴を示す永続的な情報です。
- MoReference：MO のパス参照を提供するアノテーション。

REST API を公開するには、REST API サポート用の新しい MO を作成してコネクタに登録します。これにより、開発者はそれぞれの MO エンティティに登録された REST API を Cisco UCS Director GUI で表示することができます。開発者は、この REST API を使用してコネクタで CRUD 操作を実行できます。

各 API は、**REST API ブラウザ**または **REST クライアント**を使用して実行できます。詳細な手順については、『Cisco UCS Director REST API Getting Started Guide』を参照してください。

エンティティの特定

機能とオブジェクトのレポートから REST API サポート用の MO を特定します。

たとえば、特定の物理データセンター内のすべてのクラウドアカウント、または特定のクラウドで作成されたすべての VDC を取得することができます。したがって、データセンター、クラウドアカウント、および VDC は、取得できるオブジェクトとしてモデル化されるエンティティの一部です。

REST API サポート用 POJO クラスの設定

ステップ 1 POJO クラスごとに `TaskConfigIf` インターフェイスを実装し、POJO クラスが REST 経由で公開されていることを示します。

ステップ 2 `XmlRootElement` タグを使用して各 POJO に注釈を付けます。

（注） POJO 名が MO リソース パスで POJO に対して定義されている名前と同じであることを確認してください。次に例を示します。

```
@XmlRootElement(name="foo")
```

ステップ 3 `@MoReference` タグを使用して、各 POJO の ID フィールドに注釈を付けます。

次に例を示します。

```
(@MoReference(path="foo.ID.account.ID.sample)
```

ステップ 4 MO 定義ファイル（`<featurename>-api.mo` ファイル）と、`<featurename>-url-mapping.properties` という名前のプロパティ ファイルに POJO パスを指定します。次に例を示します。

```
foo=foo.*.account.*.sample.*
```

このパスは、ツリー階層内の POJO の場所を指定します（例：`datacenter.ID`、`cloud.ID`、`vdc.ID`）。

（注） キーワードにはキャメルケースを使用できます。特殊文字は使用しないでください。

ステップ 5 MO の API リソース パス、リソース クラス、およびアダプタを含む POJO ごとに `MoPointer` を登録します。次に例を示します。

```
FooAdaptor fooAdaptor = new fooAdaptor ();
MoPointer p = new MoPointer("foo", "foo.ID.account.ID.sample",fooAdaptor, Foo.class);

parser.addMoPointer(p);
```

入力コントローラ

すべての設定ファイルには、特定のフィールドの検証および入力フォームの表示と動作の変更に使用できる *Controller* があります。

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 [UCS Manager] を選択したときに、UCS Manager のユーザ クレデンシャルを要求するプロンプトが表示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変更したりできます。
- **フォーム フィールドの検証**：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。
- **値リストの動的取得**：Cisco UCS Director データベースまたはデータ ソースから値のリストを動的に取得して、GUI フォーム オブジェクトに指定できます。

UI オブジェクトの整列化と非整列化

Controller オブジェクトは常に Config オブジェクトに関連付けられています。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、*before* と *after* の 2 つのサブステージがあります。これらの 4 つのサブステージは、Controller オブジェクトの 4 つの Action メソッドに対応しています。コントローラを使用するには、コントローラのメソッドを使用して、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）および非整列化（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：フォーム フィールドの表示と非表示の切り替え、LOV と表形式の LOV の詳細な制御に使用します。	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：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。

コントローラを使用してフィールドを検証するには、次の手順を実行します。

1. Config ファイルでアノテーションを使用して validate を `true` に設定します。
2. Controller に 1 つまたは複数の Action コールを実装します。

Config および対応する Controller の例を次に示します。

Config:

下記は UI 例の開始ページです。Config ファイルは `config` メソッドで終了し、`TaskConfigIf` インターフェイスを実装します。`@FormField` アノテーションはフロントエンドの設計に使用します。

```
@FormController(value="com.cloupia.feature.compute.api.config.controller.ComputeAccountSpecificConfigController")
@XmlRootElement(name = "ComputeAccount")
@PersistenceCapable(detachable = "true", table = "compute_account_device")
public class ComputeAccountSpecificConfig implements TaskConfigIf{
public static final String HANDLER_NAME = "Compute Account Controller";
public static final String HANDLER_LABEL = "Compute Account Controller";

public static final String TYPE_STANDARD = "0";
public static final String TYPE_ADVANCED = "1";
public static final String TYPE_CUSTOM = "2";

@Persistent
private long actionId;
@Persistent
private long configEntryId;

private int modelID;
}
```

```
// Create a controller by extending the AbstractObjectUIController.
public class ComputeAccountSpecificConfigController extends AbstractObjectUIController
{

private static Logger logger =
```

```
Logger.getLogger(ComputeAccountSpecificConfigController.class);
@Override
public void beforeMarshall(Page page, String id, ReportContext context, Object pojo)
throws Exception
{
}

}
```

Controller：

コントローラ ファイルに `<ConfigName>Controller` という名前を付けます（*<ConfigName>* は設定ファイルのベース名です）。この名前により、フレームワークはこの特定のコントローラのみを取得できます。

`AbstractObjectUIController` には、UI 入力を制御するためにオーバーライドできる 4 つのメソッドがあります。これらのメソッドは整列化と非整列化の両方の前後に呼び出され、検証に関するさまざまな機能を備えています。

整列化の前

ページ データがロードされる前に `beforeMarshall` メソッドが呼び出され、ロードされたページ データが検証されます。

例：

```
Public void beforeMarshall (Page page, String id, ReportContext context, Object pojo)
 throws Exception
{
  logger.info ("In Controller before Marshall " + context.getId());
  ComputeAccountSpecificConfig config = (ComputeAccountSpecificConfig) pojo;
  logger.info(" before Marshall ");
  page.setEmbeddedLOVs(id + ".chargeDuration", getDurationLOV());
}
```

整列化の後

ページデータがロードされた後に `afterMarshall` メソッドが呼び出されてフィールドが検証され、いくつかのフィールドが非表示になります。

例：

```
Public void afterMarshall (Page page, String id, ReportContext context, Object pojo)
 throws Exception
{
  ComputeAccountSpecificConfig config = (ComputeAccountSpecificConfig) pojo;
  page.setEditable (id + ".accountName", false);
  String accountName = context.getId();
}
```

非整列化の前

UI がロードされる前に `beforeUnmarshall` メソッドが呼び出されます。

```
Public void beforeUnmarshall (Page page, String id, ReportContext context, Object
pojo) throws Exception
{
}
```

非整列化の後

フォームの送信後に afterUnmarshall メソッドが呼び出されます。このメソッドによってページが検証されます。

例：

```
Public void afterUnmarshall(Page page, String id, ReportContext context, Object pojo)
 throws Exception
{
 ComputeAccountSpecificConfig config = (ComputeAccountSpecificConfig) pojo;
 if (page.isPageSubmitted()) {
  String accountName = config.getAccountName();
  String gateWayAddress = ipAddressBlock.getDefGw();
  String Size = ipAddressBlock.getSize();
  String toAddress = calculateIPRange(fromAddress, Size);
  if (!validateAccountName(accountName)) {
   page.setPageMessage("Invalid account name. Please use only characters. Special
characters are not allowed.");
   throw new Exception ("Invalid account name. Please use only characters. Special
characters are not allowed.");
  }
 }
}
```

ワークフロー タスクの実装

各ワークフロー タスクには、設定クラスと、タスクごとのタスク ハンドラがあります。設定クラスとタスク ハンドラは、それぞれ TaskConfigIf および AbstractTask クラスを実装します。

ワークフロータスクを実装するには、フレームワークが次の2つのシナリオに対応している必要があります。

- データベース情報の取得に使用されるフロントエンド POJO が存在している場合は、次のようなワークフロー設定クラスとそのハンドラが存在する（例：DummyAccount、DummyAccountCreateConfig.class、DummyAccountHandler.java）。
- フロントエンド POJO がない場合は、次のようなワークフロー設定クラスとそのハンドラが存在する（例：DummyAccountCreateConfig）。

ログ ファイル

API ログは、/op/infra/inframgr ディレクトリにある logfile.txt ファイルに保存されます。

ログ ファイルには、次の情報が含まれます。

- INFO（重大度キーワード）
- UTC 形式の日付/タイムスタンプ：yyyy/MM/dd-HH:mm:ss,SSS
- インストルメンテーションが実装された Java クラス

• インストルメントされたアクションの名前

発生する可能性のあるシナリオと、推奨される対処方法については、『Cisco UCS Director Open Automation Troubleshooting Guide』を参照してください。

例

例 1

次のコードスニペットにより、アダプタベースのハンドラを使用して REST API サポートを登録できます。

```
/* A REST adaptor is used to handle the CRUD operations for resources.
* You can extend the adaptor functionality by inheriting the WFTaskRestAdaptor.
* You can override the executeCustomAction, getTaskConfigImplementation, getTaskName and
 getTaskOutputDefinitions methods according to need.
*/
WFTaskRestAdaptor restAdaptor = new WFTaskRestAdaptor();
/* MoPointer is a placeholder to register the REST APIs.
* @param0 is a mandatory field, and is used to define a resource name.
* @param1 is a mandatory field, and is used to define a ResourceURL.
* @param2 is a mandatory field, and is used to define a restAdaptor.
* @param3 is a mandatory field, and is used to define the resource config class.
* If you don't want to allow the read operation for an API, use the following constructor:
* MoPointer(String name, String path, MoResourceListener moListener, Class moModel,
boolean isMoPersistent, boolean isReadAPISupported)
* mopointer must be registered in order for the API to display in REST API browser.
*/
MoPointer p = new MoPointer("ComputeAccount", "ComputeAccount", restAdaptor,
HelloWorldConfig.class);
/*
* The createOARestOperation method is used to register REST API operations through the
Open Automation connector.
* @param0 is a mandatory field, and is used to define an operation name.
* @Param1 is a mandatory field, and is used to define a resource handler name. The handler
 name is used for handling the REST API operations.
* @param2 is a mandatory field, and is used to define a resource config class. The
resource config class is required for executing a handler operation. .
*/
p.createOARestOperation("CREATE_OA", ComputeAccountCreateConfig.HANDLER_NAME,
ComputeAccountCreateConfig.class);
p.createOARestOperation("DELETE_OA", ComputeAccountDeleteConfig.HANDLER_NAME,
ComputeAccountDeleteConfig.class);
p.createOARestOperation("UPDATE_OA", ComputeAccountUpdateConfig.MODIFY_HANDLER_NAME,
ComputeAccountUpdateConfig.class);
/*
* Category is used for the REST API browser folder structure.
*/
p.setCategory(ComputeConstants.REST_API_FOLDER_NAME);
/*
* The registered REST APIs are communicated to the framework through a MoParser. Loading
 REST APIs in the framework is mandatory.
*/
parser.addMoPointer(p);
```

例 2

次のコードスニペットにより、リスナーベースのハンドラを使用して REST API サポートを登録できます。

```
/** REST Listener is used to handle the CRUD operations for the Resource.
* You can extend the Listener functionality by inheriting the AbstractResourceHandler.
* You can override the methods createResource, updateResource, deleteResource and query
 according to need.
*/
ComputeResourceAPIListner nPolcyList = new ComputeResourceAPIListner();
/* MoPointer is a placeholder to register the REST APIs.
* @param0 is a mandatory field, and is used to define a resource name.
* @param1 is a mandatory field, and is used to define a ResourceURL.
* @param2 is a mandatory field, and is used to define a Listener implementation class.
* @param3 is a mandatory field, and is used to define the resource config class. mandatory
 field.
* If you don't want to allow the read operation for an API, use the following constructor:
* MoPointer(String name, String path, MoResourceListener moListener, Class moModel,
boolean isMoPersistent, boolean isReadAPISupported)
* mopointer must be registered in order for the API to display in REST API browser.
*/
MoPointer p = new MoPointer("ComputeResource", "ComputeResource", new
ComputeResourceAPIListner, ComputeAccountListnerConfig.class);
p.setSupportedOps(MoOpType.CREATE, MoOpType.UPDATE, MoOpType.DELETE);
p.setCategory(ComputeConstants.REST_API_FOLDER_NAME);
parser.addMoPointer(p);
```

例 3

次のコードスニペットは、読み取り操作用のフロントエンドPOJOがない場合にワークフロータスク アクションの設定クラスを提供します。

```
WFTaskRestAdaptor adaptor = new WFTaskRestAdaptor();
boolean isMoPersistent = false;
p = new MoPointer("ComputeAccountConfig" ,"ComputeAccount", adaptor, null, isMoPersistent);

p.createOARestOperation("CREATE_OA", ComputeAccountCreateConfig.class);
parser.addMoPointer(p);
```

（注） パラメータをヌルで渡した場合、REST API は読み取り操作をサポートしません。

次のリストに、ここで提示したすべての例で定義されている MO の URI を示します。

- GET URI：`http://<address>/cloupia/api-v2/<name>`。`<address>` はサーバの IP アドレス、`<name>` は登録時に指定した MO ポインタ名です。

  例外：フロントエンド POJO がない設定クラスの GET URI は、Cisco UCS Director によってデフォルト動作として処理されます。

- POST URI：`http://<address>/cloupia/api-v2/<name>`。`<address>` はサーバの IP アドレス、`<name>` は登録時に指定した MO ポインタ名（例では `ComputeAccountCreateConfig`）です。設定 POJO（`ComputeAccountCreateConfig`）を HTTP リクエストの本文で XML ペイロードとして提供します

- UPDATE URI：`http://<address>/cloupia/api-v2/<name>`。`<address>` はサーバの IP アドレス、`<name>` は登録時に指定した MO ポインタ名です。HTTP リクエストの本文で設定 POJO の XML 形式を指定します。
- DELETE URI：`http://<address>/cloupia/api-v2/<name>`。`<address>` はサーバの IP アドレス、`<name>` は登録時に指定した MO ポインタ名です。HTTP リクエストの本文で設定 POJO の XML 形式を指定します。

Python スクリプトを使用した REST API の呼び出し

Cisco UCS Director REST API は、外部システムから呼び出すことができます。ここでは、Python スクリプトから REST API を呼び出す方法の例を示します。

この例では、Open Automation（OA）コンピューティングモジュール zip ファイルをアップロードして有効化した後に、Cisco UCS Director で使用できる一連の API のみを紹介します。その他の Cisco UCS Director REST API も同様の方法で呼び出すことができます。Cisco UCS Director パートナーは独自の OA モジュールを開発し、カスタム OA モジュール機能用の REST API を公開できます。カスタム OA モジュール機能用の REST API は、Cisco UCS Director サーバにアップロードして有効化してから使用できます。

こうしたカスタム OA モジュールで公開された REST API は、下記のような Python スクリプトから実行できます。

前提条件

次のスクリプトを実行するには、次の前提条件を満たす必要があります。

Cisco UCS Director サーバで、Open Automation コンピューティング モジュール `zip` ファイルをアップロードして有効にします。REST API （1 ページ）を参照してください。

クライアントで次の作業を行います。

- Python バージョン 2 リリース 2.6 以降をインストールする
- Python `requests` http ライブラリをインストールする
- Python `lxml` XML ライブラリをインストールする
- Cisco UCS Director のインストールに使用するサーバ IP アドレスと REST API アクセス キーを取得する

（注） `requests` および `lxml` ライブラリは、デフォルトの Python インストールでは使用できないため、個別にインストールする必要があります。次のスクリプトでは、`requests` を使用して HTTP REST API コールを実行し、`lxml` を使用して XML ペイロードを構築します。他のライブラリを使用してこれらのタスクを処理することもできます。

Python スクリプト

```
# This script demonstrates how to invoke the Cisco UCS Director XML REST API
#
# This script requires Python version 2, release 2.6 or later
# The Python 'requests' HTTP library is used to to invoke the REST API
# Cisco UCSD XML REST API payload and response are in XML format
# The Python 'lxml' XML library is used to construct the XML payload and to parse the
XML response
#
# Python executable

import sys
import requests
from lxml import etree
import logging

# Use the logging module to log events
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Read the UCSD server IP and REST key from the command line
try:
 number_of_arguments = len(sys.argv)
 if number_of_arguments != 3:
  raise Exception("Invalid number of arguments !! Please run script as 'python
script_file_name.py ipAddress UCSD_REST_KEY'")
except Exception as e:
 logger.error("Exception occurred while executing this script : " +str(e))
 sys.exit(1)
# IP address of UCSD server on which the REST API is to be invoked
IP_address = sys.argv[1]

# Headers for HTTP requests
headers = {}

# Authenticate the user with the REST key
rest_key_value = sys.argv[2]
content_type = "application/xml"
headers["X-Cloupia-Request-Key"] = rest_key_value
headers["content-type"] = content_type

custom_operation_type = None
resource_URL = None
http_request_type = None
resource_complete_URL = None
response = None
payload_data = None

def main():
 # Invoke the HTTP POST REST API operation 'CREATE_COMPUTE_ACCOUNT' to create a
ComputeAccount
 # custom_operation_type = CREATE_COMPUTE_ACCOUNT,
 # resource_URL = /cloupia/api-v2/ComputeAccount, HTTP request type = POST

 custom_operation_type = "CREATE_COMPUTE_ACCOUNT"
 resource_URL = "/cloupia/api-v2/ComputeAccount"
 http_request_type = "POST"
 resource_complete_URL = "https://" + IP_address + resource_URL

 # Construct XML Payload for CREATE_COMPUTE_ACCOUNT
 cuic_operation_request = etree.Element('cuicOperationRequest')
 operation_type = etree.SubElement(cuic_operation_request, 'operationType')
 operation_type.text = custom_operation_type
```

```
 payload = etree.SubElement(cuic_operation_request, 'payload')
 compute_account = etree.Element('ComputeAccount')
 account_name = etree.SubElement(compute_account, 'accountName')
 account_name.text = 'sdk_compute'
 status = etree.SubElement(compute_account, 'status')
 status.text = 'On'
 ip = etree.SubElement(compute_account, 'ip')
 ip.text = '1.1.1.1'
 inner_text = etree.tostring(compute_account)
 payload.text = etree.CDATA(inner_text)
 payload_data = etree.tostring(cuic_operation_request)

 logger.info("Executing HTTP POST CREATE_COMPUTE_ACCOUNT REST API...")
 logger.info("payload = %s",payload_data)
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP GET REST API 'Read' operation to read all ComputeAccount
 # resource_URL = /cloupia/api-v2/ComputeAccount, HTTP request type = GET

 resource_URL = "/cloupia/api-v2/ComputeAccount"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None
 logger.info("Executing HTTP GET REST API to read all ComputeAccount resource ...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeAccount
 # resource_URL = /cloupia/api-v2/ComputeAccount/{accountName}, HTTP request type = GET
 resource_URL = "/cloupia/api-v2/ComputeAccount/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None
 logger.info("Executing HTTP GET REST API to read a specific ComputeAccount resource
...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP POST REST API operation 'UPDATE_COMPUTE_ACCOUNT' to update a
ComputeAccount
 # custom_operation_type = UPDATE_COMPUTE_ACCOUNT,
 # resource_URL = /cloupia/api-v2/ComputeAccount, HTTP request type = POST

 custom_operation_type = "UPDATE_COMPUTE_ACCOUNT"
 resource_URL = "/cloupia/api-v2/ComputeAccount"
 http_request_type = "POST"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 # Construct XML Payload for UPDATE_COMPUTE_ACCOUNT
 cuic_operation_request = etree.Element('cuicOperationRequest')
 operation_type = etree.SubElement(cuic_operation_request, 'operationType')
 operation_type.text = custom_operation_type
 payload = etree.SubElement(cuic_operation_request, 'payload')
```

```
 compute_account = etree.Element('ComputeAccount')
 account_name = etree.SubElement(compute_account, 'accountName')
 account_name.text = 'sdk_compute'
 status = etree.SubElement(compute_account, 'status')
 status.text = 'Off'
 ip = etree.SubElement(compute_account, 'ip')
 ip.text = '2.2.2.2'
 inner_text = etree.tostring(compute_account)
 payload.text = etree.CDATA(inner_text)
 payload_data = etree.tostring(cuic_operation_request)

 logger.info("Executing HTTP POST UPDATE_COMPUTE_ACCOUNT REST API...")
 logger.info("payload = %s",payload_data)
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Verify the ComputeAccount updated with new ip address and status
 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeAccount
 # resource_URL = /cloupia/api-v2/ComputeAccount/{accountName}, HTTP request type = GET

 resource_URL = "/cloupia/api-v2/ComputeAccount/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 logger.info("Executing HTTP GET REST API to read a specific ComputeAccount resource
...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP POST REST API operation 'DELETE_COMPUTE_ACCOUNT' to delete a
ComputeAccount
 # custom_operation_type = DELETE_COMPUTE_ACCOUNT,
 # resource_URL = /cloupia/api-v2/ComputeAccount, HTTP request type = POST

 custom_operation_type = "DELETE_COMPUTE_ACCOUNT"
 resource_URL = "/cloupia/api-v2/ComputeAccount"
 http_request_type = "POST"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 # Construct XML Payload for DELETE_COMPUTE_ACCOUNT

 cuic_operation_request = etree.Element('cuicOperationRequest')
 operation_type = etree.SubElement(cuic_operation_request, 'operationType')
 operation_type.text = custom_operation_type
 payload = etree.SubElement(cuic_operation_request, 'payload')
 compute_account = etree.Element('ComputeAccount')
 account_name = etree.SubElement(compute_account, 'accountName')
 account_name.text = 'sdk_compute'
 inner_text = etree.tostring(compute_account)
 payload.text = etree.CDATA(inner_text)
 payload_data = etree.tostring(cuic_operation_request)

 logger.info("Executing HTTP POST DELETE_COMPUTE_ACCOUNT REST API...")
 logger.info("payload = %s",payload_data)
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
```

```
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Verify the deletion of ComputeAccount
 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeAccount
 # resource_URL = /cloupia/api-v2/ComputeAccount/{accountName}, HTTP request type = GET

 resource_URL = "/cloupia/api-v2/ComputeAccount/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 logger.info("Executing HTTP GET REST API to read a specific ComputeAccount resource
...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP POST REST API operation 'CREATE' to create a ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource, HTTP request type = POST

 resource_URL = "/cloupia/api-v2/ComputeResource"
 http_request_type = "POST"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 # Construct XML Payload for ComputeResource@CREATE

 cuic_operation_request = etree.Element('cuicOperationRequest')
 payload = etree.SubElement(cuic_operation_request, 'payload')
 compute_resource = etree.Element('ComputeResource')
 account_Name = etree.SubElement(compute_resource, 'accountName')
 account_Name.text = 'sdk_compute'
 status = etree.SubElement(compute_resource, 'status')
 status.text = 'On'
 ip = etree.SubElement(compute_resource, 'ip')
 ip.text = '1.1.1.1'
 inner_text = etree.tostring(compute_resource)
 payload.text = etree.CDATA(inner_text)
 payload_data = etree.tostring(cuic_operation_request)

 logger.info("Executing HTTP POST ComputeResource@CREATE REST API...")
 logger.info("payload = %s",payload_data)
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Verify the creation of ComputeResource
 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource/{accountName}, HTTP request type = GET


 resource_URL = "/cloupia/api-v2/ComputeResource/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 logger.info("Executing HTTP GET REST API to read a specific ComputeResource resource
...")
```

```
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP PUT REST API operation 'UPDATE' to update a ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource/{accountName}, HTTP request type =
UPDATE
 resource_URL = "/cloupia/api-v2/ComputeResource/"+"sdk_compute"
 http_request_type = "PUT"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 # Construct XML Payload for ComputeResource@UPDATE

 cuic_operation_request = etree.Element('cuicOperationRequest')
 payload = etree.SubElement(cuic_operation_request, 'payload')
 compute_resource = etree.Element('ComputeResource')
 account_name = etree.SubElement(compute_resource, 'accountName')
 account_name.text = 'sdk_compute'
 status = etree.SubElement(compute_resource, 'status')
 status.text = 'Off'
 ip = etree.SubElement(compute_resource, 'ip')
 ip.text = '2.2.2.2'
 inner_text = etree.tostring(compute_resource)
 payload.text = etree.CDATA(inner_text)
 payload_data = etree.tostring(cuic_operation_request)

 logger.info("Executing HTTP UPDATE ComputeResource@UPDATE REST API...")
 logger.info("payload = %s",payload_data)
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Verify ComputeResource account updated with new ip address and status
 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource/{accountName}, HTTP request type = GET


 resource_URL = "/cloupia/api-v2/ComputeResource/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 logger.info("Executing HTTP GET REST API to read a specific ComputeResource resource
...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Invoke the HTTP DELETE REST API Read operation - DELETE a specific ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource/{accountName}, HTTP request type =
DELETE

 resource_URL = "/cloupia/api-v2/ComputeResource/"+"sdk_compute"
 http_request_type = "DELETE"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None
```

```
 logger.info("Executing HTTP DELETE REST API to delete a specific ComputeResource resource
 ...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

 # Verify the deletion of ComputeResource 'sdk_compute'
 # Invoke the HTTP GET REST API 'Read' operation to read a specific ComputeResource
 # resource_URL = /cloupia/api-v2/ComputeResource/{accountName}, HTTP request type = GET

 resource_URL = "/cloupia/api-v2/ComputeResource/"+"sdk_compute"
 http_request_type = "GET"
 resource_complete_URL = "https://" + IP_address + resource_URL
 payload_data = None
 response = None

 logger.info("Executing HTTP GET REST API to read a specific ComputeResource resource
...")
 response = invoke_rest_API(resource_complete_URL, payload_data, http_request_type)
 logger.info("API Response HTTP Status Code = %s", response.status_code);
 logger.info("Response content type = %s", response.headers['content-type'])
 logger.info("Response content = %s", response._content)

def invoke_rest_API(complete_URL, payload_data, http_request_type):
 requests.packages.urllib3.disable_warnings()
 response = None
 if http_request_type == "GET":
  response = requests.get(complete_URL, headers=headers, verify=False)
 elif http_request_type == "POST":
  response = requests.post(complete_URL, data=payload_data, headers=headers, verify=False)

 elif http_request_type == "PUT":
  response = requests.put(complete_URL, data=payload_data, headers=headers, verify=False)

 elif http_request_type == "DELETE":
  response = requests.delete(complete_URL, headers=headers, verify=False)
 else:
  raise Exception("Invalid HTTP request type")
 return response

if __name__ == "__main__":
 try:
  main ()
 except Exception as e:
  logger.error("Exception occurred while executing this script : " +str(e))
```

コマンド ラインからスクリプトを実行する

次のように指定してコマンド ラインからスクリプトを実行します。

```
python scriptfile.py ip_address UCSD_REST_KEY
```

次に例を示します。

```
python execute_UCSD_REST_API.py 172.29.110.222 111F3D780A424C73A1C60BDD65BABB0B
```