

VMware vRealize Orchestrator 用の Web サービス クライアントの開発

vRealize Orchestrator 7.4



vmware®

最新の技術ドキュメントは VMware の Web サイト (<https://docs.vmware.com/jp/>) にあります
このドキュメントに関するご意見および感想がある場合は、docfeedback@vmware.com までお送りください。

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

ヴィエムウェア株式会社
105-0013 東京都港区浜松町 1-30-5
浜松町スクエア 13F
www.vmware.com/jp

Copyright © 2008–2018 VMware, Inc. 無断転載を禁ず。 [著作権および商標情報](#)。

目次

VMware vRealize Orchestrator 用の Web サービス クライアントの開発 5

1 Web サービス クライアントの開発 6

2 vRealize Orchestrator REST API の使用 7

Orchestrator とサードパーティ システムに対する認証 8

Orchestrator REST API を使用した vCenter Single Sign-On 認証の使用 8

Orchestrator REST API のリファレンス ドキュメントへのアクセス 13

Java REST SDK の使用 13

ワークフローを使用した操作 14

ワークフローを検索してワークフローの定義を取得する 15

ワークフローの実行 18

ワークフローのプレゼンテーションに対して入力パラメータを検証してからワークフローを実行する 20

実行中のワークフローに対するユーザー操作 24

ワークフロー操作の取得 31

ワークフローのスキーマへのアクセス 31

タスクの操作 32

タスクの作成 32

タスクの変更 33

タスクの状態の確認 34

Orchestrator インベントリでのオブジェクトの検索 35

タイプと ID によるオブジェクトの検索 35

関係を使用したオブジェクトの検索 36

フィルタの適用 37

Orchestrator オブジェクトのインポートとエクスポート 38

ワークフローのインポート 38

ワークフローのエクスポート 39

アクションのインポート 39

アクションのエクスポート 40

パッケージのインポート 40

パッケージのエクスポート 41

リソースのインポート 42

リソースのエクスポート 42

構成要素のインポート 43

構成要素のエクスポート 43

Orchestrator オブジェクトの削除 44

ワークフローの削除 44

アクションの削除 44

パッケージの削除	45
リソースの削除	45
構成要素の削除	46
Orchestrator オブジェクトに対する権限の設定	46
REST API 権限	46
ワークフローの権限の取得	47
ワークフローの権限の削除	47
ワークフローの権限の設定	48
アクションの権限の取得	48
アクションの権限の削除	48
アクションの権限の設定	49
パッケージの権限の取得	49
パッケージの権限の削除	50
パッケージの権限の設定	50
リソースの権限の取得	51
リソースの権限の削除	51
リソースの権限の設定	51
構成要素の権限の取得	52
構成要素の権限の削除	52
構成要素の権限の構成	53
プラグインを使用した操作の実行	53
プラグインに関する情報の取得	53
プラグインのインポート	54
プラグインのエクスポート	54
プラグインの有効と無効の切り替え	55
サーバ構成操作の実行	55
Orchestrator サーバ構成に関する情報の取得	55
Orchestrator サーバ構成のインポート	55
Orchestrator サーバ構成のエクスポート	56
タグ付け操作の実行	56
オブジェクトのタグ付け	57
オブジェクトのタグの解除	57
オブジェクト タグの一覧表示	58
タイプを基準にしたタグ付きオブジェクトの一覧表示	58
タグ所有者の一覧表示	58
ユーザーを基準にしたタグの一覧表示	59
タグ名でフィルタリングされたユーザーを基準にしたタグの一覧表示	59
ユーザーを基準にしたタグの削除	60

VMware vRealize Orchestrator 用の Web サービス クライアントの開発

この「VMware vRealize Orchestrator 用の Web サービス クライアントの開発」では、VMware[®] vRealize Orchestrator 用の Web サービス クライアントを開発する方法について説明します。

Orchestrator には Web サービス API が付属しているため、Web 経由でワークフローにアクセスして使用するためのアプリケーションを開発することができます。また、ワークフローに対してさまざまな操作を実行するための Representational State Transfer (REST) API も付属しています。

対象者

ここに記載されている情報は、RESTful Web サービスを使用してネットワーク上の Orchestrator プロセスにアクセスする必要があるアプリケーション開発者向けの情報です。

Web サービス クライアントの開発

VMware vRealize Orchestrator には Web サービス API が付属しているため、Web 経由でワークフローにアクセスするためのアプリケーションを開発することができます。Orchestrator Web サービス API の主な目的は、カスタムの Web ベース アプリケーションに Orchestrator ワークフローを統合できるようにすることです。

Orchestrator では REST (Representational State Transfer) API に基づく Web サービス API が提供されます。Orchestrator REST API は、Orchestrator サーバのインベントリやインストール済みプラグインのインベントリのオブジェクトを、事前定義されている URL でアクセスできるリソースとして公開します。これらの URL に HTTP 要求を行うと、ワークフローを介して操作がトリガーされます。Orchestrator REST API は、RESTful Web サービスのセットを介してインベントリ オブジェクトをリソースとして公開します。このサービスを使用して、ワークフローの定義の取得、ワークフローの実行、実行中のワークフローのステータスの確認、ワークフロー実行のキャンセル、待機中のユーザー対話の処理、ワークフローのプレゼンテーションの取得などを実行できます。

vRealize Orchestrator REST API の使用

Orchestrator REST API は HTTP を介して Orchestrator サーバと直接通信し、ワークフローに対してさまざまな操作を実行するための機能を提供します。

Orchestrator REST API は Orchestrator サーバのインベントリやインストール済みプラグインのオブジェクトを、事前定義されている URL にリソースとして公開します。これらの URL の HTTP 呼び出しを行って Orchestrator で操作を起動します。これにより、ワークフローに対して次のさまざまなタスクを実行できます。

- ワークフローの実行、ワークフローのスケジュール設定、ワークフローの実行の取得、ユーザー操作に対する応答、ワークフローの実行のキャンセル。
- 入力パラメータ、出力パラメータ、それらのプレゼンテーションなど、ワークフローに関する詳細の取得。
- 状態、生成ログ、開始日、終了日など、ワークフローの実行に関する詳細の取得。
- Orchestrator やインストール済みプラグインのインベントリの参照。
- ワークフロー、アクション、パッケージのインポートおよびエクスポート。

Orchestrator REST API を使用することにより、任意のプログラミング言語でビルド可能なカスタム アプリケーションに Orchestrator ワークフローを簡単に統合できます。

また、Orchestrator REST API は eTag をサポートするほか、応答データをキャッシュするメカニズムも提供します。

この章では次のトピックについて説明します。

- [Orchestrator とサードパーティ システムに対する認証](#)
- [Orchestrator REST API のリファレンス ドキュメントへのアクセス](#)
- [Java REST SDK の使用](#)
- [ワークフローを使用した操作](#)
- [タスクの操作](#)
- [Orchestrator インベントリでのオブジェクトの検索](#)
- [Orchestrator オブジェクトのインポートとエクスポート](#)
- [Orchestrator オブジェクトの削除](#)
- [Orchestrator オブジェクトに対する権限の設定](#)
- [プラグインを使用した操作の実行](#)

- サーバ構成操作の実行
- タグ付け操作の実行

Orchestrator とサードパーティ システムに対する認証

Orchestrator REST API を使用して作成した HTTP 要求内で、Orchestrator に対する認証を行う必要があります。Orchestrator REST API を使用して vCenter Server や vRealize Automation などのサードパーティ システム上のリソースにアクセスする場合は、そのシステムに対する認証も行う必要があります。

たとえば、Orchestrator インベントリ内のすべてのワークフローにアクセスするには、Orchestrator に対する認証を行う必要があります。ただし、vCenter Server に対してワークフローを実行する場合は、Orchestrator と vCenter Server に対して認証を行う必要があります。

vRealize Automation と vSphere のどちらを使用して Orchestrator を認証プロバイダとして構成するかによって、Orchestrator REST API の認証スキームは異なります。Orchestrator で vCenter Single Sign-On を使用する場合は、現在の構成内容に応じて、vCenter Single Sign-On サーバから発行された holder-of-key トークンを使用して認証を行うことができます。Orchestrator が vRealize Automation で構成されている場合、OAuth ベアラー アクセス トークンを使用して認証を行うことができます。

Orchestrator REST API の最上位 URL で HTTP 要求を作成する場合は、Orchestrator に対して認証を行う必要はありません。Orchestrator REST API の最上位 URL は、`https://<orchestrator_host>:<port>/vco/api/` です。

注意 外部 Orchestrator のデフォルト ポート番号は 8281 です。vRealize Automation に組み込まれている Orchestrator インスタンスのデフォルト ポート番号は 443 です。

REST API の最上位 URL の GET 要求は、REST API 経由でアクセス可能なすべてのリソースに対する URL を返します。これらの URL で HTTP 要求を作成するには、Orchestrator に対して認証を行う必要があります。

Orchestrator REST API を使用した vCenter Single Sign-On 認証の使用

vSphere 認証モードを使用して Orchestrator に vCenter Single Sign-On サーバが構成されている場合、Orchestrator REST API を使用して Orchestrator のシステム オブジェクトにアクセスするには、holder-of-key プリンシパル トークンが必要になります。Orchestrator サーバを介して vCenter Single Sign-On サーバを使用している vCenter Server またはサードパーティ システムにアクセスするには、Orchestrator 用の holder-of-key 委任 トークンと、ユーザーの holder-of-key プリンシパル トークンが必要になります。

Orchestrator に vCenter Single Sign-On サーバが構成されている場合、有効な認証情報を使用して認証を行うと、Orchestrator は holder-of-key トークンを管理します。

Orchestrator のシステム オブジェクトへのアクセス

Orchestrator のシステム オブジェクトには、REST API のインベントリ サービスおよびカタログ サービスの URL からアクセスできます。

- `https://<orchestrator_host>:<port>/vco/api/inventory/System/`
- `https://<orchestrator_host>:<port>/vco/api/catalog/System/`

Orchestrator のシステム オブジェクトにアクセスする際、インベントリ サービスまたはカタログ サービス用に作成した HTTP 要求の **Authorization** ヘッダーに holder-of-key プリンシパル トークンを渡します。

たとえば、タイプが **Workflow** であるすべてのシステム オブジェクトを取得するには、

`https://<orchestrator_host>:<port>/vco/api/catalog/System/Workflow/` から **GET** 要求を行います。

Orchestrator に対して認証を行うには、要求の **Authorization** ヘッダーにユーザーの holder-of-key プリンシパル トークンを渡す必要があります。

サードパーティ システムのオブジェクトへのアクセス

Orchestrator REST API を使用して vCenter Single Sign-On サーバに登録されているサードパーティ システムで操作を実行するには、Orchestrator とサードパーティ システムに対して認証を行う必要があります。Orchestrator REST API を使用して作成する HTTP 呼び出しに 2 つのヘッダーを含めます。

- **Authorization**.このヘッダーにユーザーの holder-of-key プリンシパル トークンを渡す必要があります。
- **VCOAuthorization**.このヘッダーに Orchestrator の holder-of-key 委任トークンを渡す必要があります。Orchestrator の委任トークンは vCenter Single Sign-On サーバから取得する必要があります。Orchestrator は委任トークンを使用してユーザーに代わってサードパーティ システムに対する認証を行います。

たとえば、Orchestrator REST API を使用して仮想マシンを使用しているワークフローを実行するには、Orchestrator と vCenter Server の両方からリソースにアクセスします。Orchestrator と vCenter Server に対して認証を行うには、作成した要求の **Authorization** ヘッダーに holder-of-key プリンシパル トークンを渡し、

VCOAuthorization ヘッダーに委任トークンを渡す必要があります。これにより、ユーザーはプリンシパル トークンを使用して Orchestrator に対する認証を行い、Orchestrator は委任トークンを使用してユーザーの代わりに vCenter Server に対する認証を行います。

vCenter Single Sign-On サーバは Orchestrator をソリューションとして処理します。すべてのソリューションは一意のユーザー名を使用して vCenter Single Sign-On サーバに登録されます。Orchestrator のソリューション ユーザー名と vCenter Single Sign-On サーバの holder-of-key プリンシパル トークンを vCenter Single Sign-On サーバに渡して、Orchestrator の委任トークンを要求します。vCenter Single Sign-On サーバが発行するトークンは、Orchestrator がユーザーに代わってサードパーティ システムに対する認証を行うための holder-of-key token 委任トークンです。

例: vCenter Single Sign-On モードでのセッションの取得

次のコード例では vCenter Single Sign-On モードでセッションを取得しています。

```
URI uri = URI.create("https://orchestrator-server:8281/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

//provide the address of the vCenter Single Sign-On server
URI ssoUri = URI.create("https://sso-server:7444/ims/STSService?wsdl");

//set the tokens to be valid for an hour
long lifeTimeSeconds = 60 * 60;

//create a factory for vCenter Single Sign-On tokens
SsoAuthenticator sso = new SsoAuthenticator(URI ssoUri, URI adminUri, VcoSessionFactory
vcoSessionFactory, long lifeTimeSeconds);
```

```
//provide vCenter Single Sign-On credentials
SsoAuthentication authentication = sso.createSsoAuthentication("username", "password");

VcoSession session = sessionFactory.newSession(authentication);
//use session here
```

Orchestrator のソリューション ユーザー名の取得

vCenter Single Sign-On サーバは Orchestrator をソリューションとして処理します。すべてのソリューションは一意のユーザー名を使用して vCenter Single Sign-On サーバに登録されます。vCenter Single Sign-On サーバから Orchestrator 用の holder-of-key 委任トークンを要求できるようにするには、Orchestrator のソリューション ユーザー名が必要です。

開始する前に

vCenter Single Sign-On サーバが発行した有効なプリンシパルの holder-of-key トークンがあることを確認します。

手順

- 1 Orchestrator のソリューション ユーザー名の URL で以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/users/
```

- 2 要求の **Authorization** ヘッダーにプリンシパル holder-of-key トークンを指定します。

要求の `<user solution-user="<OrchestratorSolutionUserName>" />` 要素には、Orchestrator のソリューション ユーザー名が格納されます。次に、Orchestrator のソリューション ユーザー名の例を示します。

```
<user xmlns="http://www.vmware.com/vco" admin-rights="true" solution-
user="vCO-15d98795afa5b0d6f47ee3aeab3">
```

次に進む前に

Orchestrator のソリューション ユーザー名とプリンシパルの holder-of-key トークンを使用して、vCenter Single Sign-On サーバから holder-of-key 委任トークンを要求します。

構成済みの vRealize Automation 認証での vRealize Orchestrator REST API SDK の使用

マルチテナントまたは単一テナント環境の構成済み vRA 認証では、REST API SDK を使用できます。

以下のコードに必要な認証トークン (OAuth2.0) を取得するには、ナレッジベースの記事「[vRO REST API authorization using OAuth2.0 Authentication](#)」(KB2148518) を参照してください。

注意 vRealize Automation 認証モードでのセッションの取得

次のコード例は、単一テナントとマルチテナント環境の両方で、vRealize Automation 認証モードでセッションを取得します。

- マルチテナントが有効になっていない場合：

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTRiM2UtYmYyYi04ZmRmNzY3N"
+
"GZiZWElCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUeHfUKUuTE9DQUwiLCJkb21haw4iOiJ2c3BoZXJlLmxvY2FsIiwidXNlcl9pZCI6Ij"
+
"MiLCJhdXR0X3RpbWUiOiJlMDIyMDIxMTAsImZcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+
"vbS9TQUFTL3QvdnNwaGVyZS5sb2Nhbc9hdXR0IiwiaXVkiOiJjaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbWJlLmVuZy52bXdhcmUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvd2F1dGh0b2t1biIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpvYXNpZcpuYW1lc2p"
+
"0YzptQU1MOjIuMDphYzpjbjGFzc2VzO1Bhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNTAyMjAyMTEwLW1lc2p"
+
"n1dIiwic2NwIjojdXNlciIsImZcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lbmcudm13YXJlLnN"
+
"td2FyZS5jb20iLCJjaWQ0IiwiaXVkiOiJhZG1pbmlzdHJhdG9yQHNmLTl5LTl5LnNvZi1tYnUuZW5nLnZ"
+
"td2FyZS5jb20iLCJjaWQ0IiwiaXVkiOiJhZG1pbmlzdHJhdG9yQHNmLTl5LTl5LnNvZi1tYnUuZW5nLnZ"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU0WlxiOGUxNyIsInByb190eXB1Ijo"
+
"iVFNlbnN1IiwiaXVkiOiJhZG1pbmlzdHJhdG9yQHNmLTl5LTl5LnNvZi1tYnUuZW5nLnZ"
+
"G9gEQPtmeH5jYab-IlTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-qqKutZL21R"
+
"m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//Use the session here
```

- マルチテナントが有効な場合：
- 通常のテナントのユーザーの場合：

```
URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);
String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJjNzg4NWNiYS1hZTFmLTRiM2UtYmYyYi04ZmRmNzY3N"
+
"
```

```

"GZiZWEiLCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUehFUKUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxv
Y2FsIiwidXNlcl9pZCI6Ij"
+
"MiLCJhdXR0X3RpbWUiOjE1MDIyMDIxMTAsImZcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lb
mcudm13YXJlLnN"
+
"vbS9TQUFTL3QvbnNwaGVyZS5sb2Nhbc9hdXR0IiwiaXVkaWVkaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbW
J1LmVuZy52bXdhcmUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmxvY2FsL2F1dGgvd2F1dGh0b2t1biIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpv
YXNpczpuYW1lc3p"
+
"0YzpTQU1M0jIuMDphYzpjbgGFzc2VzOlBhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNT
AyMjAyMTEwLW1lc3p"
+
"n1dIiwic2NwIjoidXNlciIsImZcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lb
mcudm13YXJlLnN"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEEx6bURnIiwiaXVkaWVkaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbW
J1LmVuZy52bXdhcmUuY29tL"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEEx6bURnIiwiaXVkaWVkaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbW
J1LmVuZy52bXdhcmUuY29tL"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybl90e
XB1IjoiVVNFUiJ9."
+ "G9gEQPtMEH5jYab-
ILTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-
qgKutZl21R"
+ "m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

VcoSession session = sessionFactory.newSession(auth);
//The operations will be executed in the scope of the tenant authenticated with the
token above.

//Use the session below

```

■ ソリューション ユーザーの場合：

ソリューション ユーザーは、自身のテナントの範囲と、通常のテナントの範囲内で作業できます。ソリューション ユーザーは、実行する操作の範囲をオーバーライドできます。

```

URI uri = URI.create("https://orchestrator-server:8283/vco/api");
VcoSessionFactory sessionFactory = new DefaultVcoSessionFactory(uri);

example

String token =
"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJqdGkiOiJhZG1pbmlzdHJhdG9yQFZTUehFUKUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxv
Y2FsIiwidXNlcl9pZCI6Ij"
+
"GZiZWEiLCJwcm4iOiJhZG1pbmlzdHJhdG9yQFZTUehFUKUuTE9DQUwiLCJkb21haW4iOiJ2c3BoZXJlLmxv
Y2FsIiwidXNlcl9pZCI6Ij"
+

```

```

"MiLCJhdXR0X3RpbWUiOjE1MDIyMDIxMTAsImIzcyI6Imh0dHBzOi8vc2YtMjktMTAtMjkuc29mLW1idS5lb
mcudm13YXJlLnN"
+
"vbS9TQUFTL3QvbnNwaGVyZS5sb2Nhbc9hdXR0IiwiaXVvIjoiaHR0cHM6Ly9zZi0yOS0xMC0yOS5zb2YtbW
J1LmVuZy52bXdhcmUuY29tL"
+
"1NBQVMvdC92c3BoZXJlLmXvY2FsL2F1dGgvd2F1dGh0b2t1biIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpv
YXNpczpuYW1lc3p"
+
"0YzpTQU1M0jIuMDphYzpjbgFzc2VzO1Bhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNT
AyMjAyMTEwLW1lc3p"
+
"n1dIiwic2NwIjoiaXNlciIsImIkcCI6IjAiLCJlbWwiOiJhZG1pbmlzdHJhdG9yQHNMLTl5LTl5LnNv
Zi1tYnUuZW5nLnZ"
+
"td2FyZS5jb20iLCJjaWQiOiJjYWZlX2NsaS1yRlJlNEx6bURnIiwiaXVvIjoiaXVvIiwid2lkIjoiaXVvIiwiaXVvIj
oxNTAyMjAyMTEwLW1lc3p"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInBybl90e
XB1IjoiaXVvIiwiaXVvIj9."
+ "G9gEQPtMEH5jYab-
ILTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-
qgKutZl21R"
+ "m740qBKLhmBB0NQg19ysMAVJNSxapFzirmWurF_5CKpv4WM7Y8H_bY9iNmDKQTXI";

//provide OAuth2 token obtained in step 1 here
Authentication auth = new OAuthTokenAuthentication(token);

// By default each tenant works in its tenant scope. However, solution users can
// overrde the tenant in which they perform a given operation:
// Here, users of SDK should provide a value that is meaningful to their context.
String overrideWithTenant = "<nonSolutionUserTenant>";

VcoSession session = sessionFactory.newSession(auth, overrideWithTenant);

//Use session below

```

Orchestrator REST API のリファレンス ドキュメントへのアクセス

Orchestrator REST API のリファレンス ドキュメントには、API の RESTful Web サービスに関する情報や、API で使用可能なデータ モデルと応答コードに関する情報などのほかに、サンプル コードも記載されています。

Orchestrator REST API のリファレンス ドキュメントは、Orchestrator とともにインストールされます。リファレンス ドキュメントは、https://<orchestrator_host>:<port>/vco/api/docs/ から入手することもできます。

正式な Swagger の仕様は、<https://swagger.io/specification/> で確認できます。

Java REST SDK の使用

Java SDK ライブラリを使用して Java アプリケーションの Orchestrator REST API の操作を呼び出し、オブジェクトを直接操作できます。

Orchestrator REST SDK のすべての RESTful Web サービスは、サービスの使用により実行可能な操作に対応するメソッドを持つ Java ラッパー クラスを持っています。

Java REST SDK は Orchestrator とともにインストールされます。Java REST SDK アーティファクトは次の場所から入手できます。

注意 Orchestrator Appliance を展開している場合は、アーティファクトにのみアクセスできます。

- https://<orchestrator_host>:<port>/vco-repo/com/vmware/o11n/o11n-rest-client/
- https://<orchestrator_host>:<port>/vco-repo/com/vmware/o11n/o11n-rest-client-examples/
- https://<orchestrator_host>:<port>/vco-repo/com/vmware/o11n/o11n-rest-client-services/
- https://<orchestrator_host>:<port>/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/

例: ワークフローの実行および完了の待機

次のコード例ではワークフローを実行し、完了するのを待機しています。

```
//start a new session to Orchestrator by using specified credentials
VcoSession session = DefaultVcoSessionFactory.newLdapSession(new URI("https://orchestrator-
server:8281/vco/api/"), "username", "password");

//create the services
WorkflowService workflowService = new WorkflowService(session);
ExecutionService executionService = new ExecutionService(session);

//find a workflow by ID
Workflow workflow = workflowService.getWorkflow("1231235");

//create an ExecutionContext from the user's input
ExecutionContext context = new ExecutionContextBuilder().addParam("name",
"Jerry").addParam("age", 18).build();

//run the workflow
WorkflowExecution execution = executionService.execute(workflow, context);

//wait for the workflow to reach the user interaction state, checking every 500 milliseconds
execution = executionService.awaitState(execution, 500, 10, WorkflowExecutionState.CANCELED,
WorkflowExecutionState.FAILED, WorkflowExecutionState.COMPLETED);

String nameParamValue = new
ParameterExtractor().fromTheOutputOf(execution).extractString("name");
System.out.println("workflow was executed with 'name' input set to" + nameParamValue);
```

ワークフローを使用した操作

Orchestrator REST API には、ワークフローを使用した各種操作の実行に使用できる Web サービスが付属しています。

ワークフローを検索してワークフローの定義を取得する

ワークフローで任意の操作を実行するには、そのワークフローを Orchestrator インベントリで検索し、そのワークフローの定義を取得する必要があります。ワークフローの定義により、そのワークフローの入力パラメータと出力パラメータ、使用可能なワークフローへのリンク、ワークフローのプレゼンテーション、その他のオブジェクトを確認することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

1 ワークフローのインベントリ アイテムを検索します。

- ワークフローの完全な名前または名前のキーワードがわかっている場合は、ワークフロー サービスの URL で以下の **GET** 要求を作成してフィルタを指定します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows?
conditions=name={<workflowFullName>}
```

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows?
conditions=name~{<keyWord>}
```

- ワークフローのインベントリ アイテムのエントリ ポイントとなる URL で以下の **GET** 要求を作成して、カタログ サービスまたはインベントリ サービスでワークフローを検索します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/catalog/System/Workflow/
```

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/inventory/System/Workflows/
```

2 ワークフローのインベントリ アイテムの URL で以下の **GET** 要求を作成して、ワークフローのインベントリ アイテムを取得します。

```
GET https://{<orchestrator_host>}:
{<port>}/vco/api/catalog/System/Workflow/{<workflowID>}/
```

3 ワークフローの定義の URL で以下の **GET** 要求を作成して、ワークフローの定義を取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

例: 「Hello の送信」 ワークフローの検索

「Hello の送信」 ワークフローを検索し、そのワークフローの定義を取得することができます。

- 1 「Hello の送信」 ワークフローを検索するには、ワークフロー サービスの URL で以下の **GET** 要求を作成してフィルタを指定します。

```
GET https://localhost:8281/vco/api/workflows?conditions=name~Hello
```

名前に「Hello」という単語が含まれているワークフローのリストが表示されます。

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-items xmlns="http://www.vmware.com/vco" total="2">
  <link rel="down"
    href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF808080808080808080808080E680808001308666823
6014a0614d16e1/">
    <attributes>
      <attribute name="id"
value="CF808080808080808080808080E6808080013086668236014a0614d16e1"/>
      <attribute name="canExecute" value="true" />
      <attribute name="description" value="" />
      <attribute name="name" value="Interactive Hello World" />
      <attribute name="type" value="Workflow"/>
      <attribute name="canEdit" value="true"/>
    </attributes>
  </link>
  <link rel="down"
    href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF808080808080808080808080DA80808001308666823
6014a0614d16e1/">
    <attributes>
      <attribute name="id"
value="CF808080808080808080808080DA808080013086668236014a0614d16e1"/>
      <attribute name="canExecute" value="true" />
      <attribute name="description" value="" />
      <attribute name="name" value="Send Hello" />
      <attribute name="type" value="Workflow"/>
      <attribute name="canEdit" value="true"/>
    </attributes>
  </link>
</inventory-items>
```

- 2 「Hello の送信」 ワークフローのインベントリ アイテムの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:
8281/vco/api/catalog/System/Workflow/CF808080808080808080808080DA80808001308666823
6014a0614d16e1/
```


応答本文に、「Hello の送信」ワークフローのインベントリ アイテムが表示されます。

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<inventory-item xmlns="http://www.vmware.com/vco"
  href="https://localhost:
8281/vco/api/catalog/System/Workflow/CF8080808080808080808080808080DA80808001308666823
6014a0614d16e1/">
  <relations>
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/" />
    </relations>
  <attributes>
    <attribute name="id"
value="CF8080808080808080808080808080DA808080013086668236014a0614d16e1"/>
    <attribute name="canExecute" value="true" />
    <attribute name="description" value="" />
    <attribute name="name" value="Send Hello" />
    <attribute name="type" value="Workflow"/>
    <attribute name="canEdit" value="true"/>
  </attributes>
</inventory-item>
```

- 3 ワークフローの定義を取得するには、そのワークフローの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080DA808080013086668236014a061d16e  
1/
```

応答本文に、「Hello の送信」ワークフローの定義が表示されます。

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
    href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/">
    <relations>
        <link rel="up"
            href="https://localhost:
8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
        <link rel="add"
            href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
        <link rel="down"
            href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/presentation/" />
```

```
<link rel="down"  
      href="https://localhost:  
8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e  
1/tasks/" />  
      <link rel="down"  
            href="https://localhost:  
8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e  
1/icon/" />  
</relations>  
<input-parameters>  
    <parameter name="name" type="string" />  
</input-parameters>  
<output-parameters>  
    <parameter name="message" type="string" />  
</output-parameters>  
<name>Send Hello</name>  
    <description></description>  
</workflow>
```

ワークフローの実行

ワークフローを実行するには、Orchestrator REST API を使用して、特定のワークフロー用に新しい実行オブジェクトを作成します。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 定義の URL で以下の **GET** 要求を作成して、実行するワークフローの定義を取得します。

GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/

要求の応答本文に、ワークフローの定義が表示されます。ワークフローの定義では、ワークフローの入力パラメータ、ワークフローの説明、その他の情報を確認することができます。

- 2 ワークフローの実行オブジェクトが存在する URL で以下の **POST** 要求を作成します。

POST https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/executions/

- 3 要求本文内の **execution-context** 要素で、ワークフローの入力パラメータの値を指定します。

要求本文に空の **execution-context** を指定した場合、入力パラメータのデフォルト値（存在する場合）を使用してワークフローが実行されます。

POST 要求が正常に処理されると、ステータス コード 202 とともに空の応答本文と、**Location** ヘッダー内に新しく作成された実行オブジェクトへのリンクを受信します。

例: 「Hello の送信」 ワークフローの実行

「Hello の送信」ワークフローの定義を取得して実行することができます。

- 1 「Hello の送信」 ワークフローの定義が存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e  
1/
```

要求の応答本文に、ワークフローの定義が表示されます。

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
  href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/">
  <relations>
    <link rel="up"
      href="https://localhost:
8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
    <link rel="add"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/presentation/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/tasks/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e
1/icon/" />
  </relations>
  <input-parameters>
    <parameter name="name" type="string" />
  </input-parameters>
  <output-parameters>
    <parameter name="message" type="string" />
  </output-parameters>
  <name>Send Hello</name>
  <description></description>
</workflow>
```

- 2 ワークフローの実行オブジェクトが存在する URL で以下の **POST** 要求を作成します。

```
POST https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e  
1/executions/
```

要求本文内の `execution-context` 要素で、入力パラメータの値を渡します。

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

ワークフローのプレゼンテーションに対して入力パラメータを検証してからワークフローを実行する

ワークフローのプレゼンテーションにより、ワークフローの入力パラメータに渡すことができる値の制約（値の事前定義済みリスト、値の範囲など）を定義することができます。ワークフローが正常に実行することを確認するには、ワークフローのプレゼンテーションの定義に対して、ワークフローの入力パラメータに渡す値を検証する必要があります。

カスタム アプリケーションのワークフローを統合する際、場合によっては、ワークフローの実行時にワークフローの入力パラメータの値を入力するためのウィザードを作成する必要があります。ワークフローのプレゼンテーション サービスを使用することにより、ワークフローのプレゼンテーションをインスタンス化し、ウィザードの異なる画面に対応するように入力パラメータの値を分けて渡すことができます。入力パラメータに渡す値は、ワークフローのプレゼンテーションで定義されている制約に対して検証することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 ワークフローの定義が存在する URL で以下の **GET** 要求を作成して、実行するワークフローの定義を取得します。

GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/

要求の応答本文に、ワークフローの定義が表示されます。ワークフローの定義では、ワークフローの入力パラメータ、ワークフローの説明、その他の情報を確認することができます。

- 2 ワークフローの URL で以下の **GET** 要求を作成して、ワークフローのプレゼンテーションの定義を取得します。

```
GET https://{<orchestrator_host>}:
      {<port>}/vco/api/workflows/{<workflowID>}/presentation/
```


要求の応答本文に、ワークフローの定義が表示されます。

```
<xml version="1.0" encoding="UTF-8" standalone="yes">
<workflow xmlns="http://www.vmware.com/vco" customized-icon="false"
  href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/">
  <relations>
    <link rel="up"
      href="https://localhost:
8281/vco/api/inventory/System/Workflows/Samples/HelloWorld/" />
    <link rel="add"
      href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/executions/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/presentation/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/tasks/" />
    <link rel="down"
      href="https://localhost:
8281/vco/api/workflows/CF808080808080808080808080808080DA808080013086668236014a0614d16e
1/icon/" />
  </relations>
  <input-parameters>
    <parameter name="name" type="string" />
  </input-parameters>
  <output-parameters>
    <parameter name="message" type="string" />
  </output-parameters>
  <name>Send Hello</name>
  <description></description>
</workflow>
```

- 2 ワークフローのプレゼンテーションの定義が存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e  
1/presentation/
```

- 3 ワークフローのプレゼンテーションの実行インスタンスが存在する URL で以下の **POST** 要求を作成します。

```
POST https://localhost:8281/vco/api/workflows/CF80808080808080808080808080DA808080013086668236014a0614d16e1/presentation/instances/
```

空の **execution-context** を指定し、入力パラメータに値を指定しないでプレゼンテーションのインスタンス化が行われるようにします。

```
<execution-context xmlns="http://www.vmware.com/vco"/>
```

応答本文にはすべてのフィールドに適用されるエラーメッセージが記載され、入力パラメータの値が無効であることが示されます。

```

.....
<fields>
  <field type="string" hidden="false" id="name">
    <display-name>name</display-name>
    <description>name</description>
    <messages>
      <message severity="ERROR" code="VCO-CNS0002">
        <Summary>
          The minimum number of characters allowed for this field is 3.0
        </Summary>
      </message>
    </messages>
    <constraints>
      <number-range max="15.0" min="3.0" />
    </constraints>
  </field>
</fields>
.....

```

- 4 特定のプレゼンテーションのインスタンスが存在する URL で以下の **POST** 要求を作成します。

```
POST https://localhost:8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e1/presentation/instances/88808080808080808080808080803F8080800132145338690643f66a027ec/
```

要求本文で、入力パラメータの値を指定します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

要求の応答本文で、入力パラメータの値が正しいかどうかを確認します。

```
<execution started-by="vcoadmin" .... valid="true".....>
```

- 5 プレゼンテーションが正しい場合、実行中のワークフローが存在する URL で以下の **POST** 要求を作成し、ワークフローを実行します。

```
POST https://localhost:8281/vco/api/workflows/CF80808080808080808080808080DA808080013086668236014a0614d16e1/executions/
```

要求本文で、ワークフローの入力パラメータの値を渡します。ワークフローのプレゼンテーションの出力パラメータとして返された値と同じ値を使用するか、ワークフローのプレゼンテーションに対して行った最後の **POST** 要求本文を直接使用します。

実行中のワークフローに対するユーザー操作

Orchestrator REST API を使用すると、実行中のワークフローに対してさまざまな操作を実行できます。実行中のワークフローのステータスを取得したり、待機中のユーザー操作に応答したり、ワークフローの実行をキャンセルしたりすることができます。

ワークフロー実行オブジェクトの取得とワークフローステータスの確認

ワークフローの実行に関する情報（開始日と終了日、実行の状態、入力パラメータの値など）を取得できます。また、ワークフロー実行用に生成されたログを取得することもできます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 ワークフローの URL で以下の **GET** 要求を作成して、ステータスをチェックするワークフローの定義を取得します。

GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/

要求の応答本文に、ワークフローの定義が表示されます。ワークフローの定義には、そのワークフローの実行オブジェクトに対するリンクが含まれています。

- 2 URL で以下の **GET** 要求を作成して、ワークフローの使用可能な実行インスタンスを取得します。

GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/executions/

要求の応答本文に、ワークフローの使用可能な実行インスタンスがリストされます。このリストで、すべてのワークフロー実行の開始日と終了日、およびそれらのステータスと開始ユーザーを確認できます。

- 3 (オプション) ワークフローの特定の実行に関する詳細情報を取得するには、その実行の URL で以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
      {<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/
```

要求の応答本文に、特定のワークフロー実行の XML 表現が表示されます。この実行のために渡された入力パラメータの値、実行を開始したユーザー、開始日と終了日、および実行の状態を確認できます。

- 4 (オプション) ワークフロー実行について生成されたログを取得するには、ログが保持されている URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:  
{<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/logs/
```

- 5 (オプション) 実行の状態に関する追加情報を取得するには、ワークフローの状態が保持されている URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:
    {<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/state/
```

例: 「Hello の送信」 ワークフローの実行を取得し特定の実行の状態を確認する

「Hello の送信」ワークフローを実行することで、使用可能な実行オブジェクトを取得し、それらの詳細を確認することができます。

- 1 「Hello の送信」ワークフローの定義が保持されている URL で以下の **GET** 要求を作成して、ワークフローの定義を取得します。

```
GET https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080DA808080013086668236014a0614d16e  
1/
```

- 2 ワークフローの実行オブジェクトが保持されている URL で以下の **GET** 要求を作成して、ワークフローの使用可能な実行を取得します。

```
GET https://localhost:  
8281/vco/api/workflows/CF8080808080808080808080808080DA808080013086668236014a0614d16e  
1/executions/
```

- 3 要求の応答本文から、ワークフローの実行を選択し、それを取得する **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/workflows/CF808080808080808080808080DA808080013086668236014a0614d16e  
1/executions/8880808080808080808080808080803A8080800132145338690643f66a027ec/
```

応答本文には、指定された ID を持つワークフロー実行の XML 表現が含まれています。その表現で実行に関する詳細を確認できます。

```
.....
<input-parameters>
  <parameter name="name" type="string">
    <string>John Smith</string>
  </parameter>
</input-parameters>
<output-parameters>
  <parameter name="message" type="string">
    <string>Hello, John Smith!</string>
  </parameter>
</output-parameters>
<start-date>2012-01-31T14:28:40.223+03:00</start-date>
<end-date>2012-01-31T14:28:40.410+03:00</end-date>
<started-by>vcadmin</started-by>
<name>Send Hello</name>
.....
```

待機中のユーザー操作への応答

Orchestrator REST API を使用して、待機中のユーザー操作に応答することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 有効なユーザー操作オブジェクトが存在する URL で **GET** 要求を行いすべてのユーザー操作オブジェクトのリストを取得するか、または待機中のユーザー操作オブジェクトのみをフィルタリングして取得します。

URL	説明
<code>https://<orchestrator_host>:<port>/vco/api/catalog/System/UserInteraction</code>	Orchestrator の有効なユーザー操作オブジェクトが存在する URL
<code>https://<orchestrator_host>:<port>/vco/api/catalog/System/UserInteraction?status=0</code>	待機中のユーザー操作オブジェクトのみをフィルタリングする URL

有効なユーザー操作オブジェクトのリストが取得されます。待機中のユーザー操作には、値が **waiting** で **state** という名前の属性が設定されています。

- 2 応答したい待機中のユーザー操作のインベントリ アイテムが存在する URL で、以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
{<port>}/vco/api/catalog/System/UserInteraction/{<userInteractionID>}/
```

インベントリのアイテムにはユーザー操作インスタンスへのリンクが含まれています。このユーザー操作インスタンスには、特定のワークフローの実行が関連付けられています。

- 3 ユーザー操作インスタンスの URL で、特定のワークフローを実行するため、以下の **POST** 要求を作成します。

POST https://{orchestrator_host}:

{<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/interaction/

- 4 要求本文内の **execution-context** 要素で、ユーザー操作の入力パラメータの値を指定します。

REST API は、ユーザー操作に正しい応答が行われた場合、204 個のステータスを返します。

例: 対話式の「Hello World」ワークフローのユーザー操作に応答する

対話式の「Hello World」サンプルワークフローを実行して、ユーザー操作に応答します。

- 1 カタログサービスのユーザー操作オブジェクトのエンドポイントで以下の **GET** 要求を作成して、ワークフローの待機中のユーザー操作を検索します。

```
GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0
```

- 2 対話式の「Hello World」ワークフローのユーザー操作インベントリ オブジェクトを探し、そのオブジェクトの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/8880808080808080808080805A8080800132
145338690643f66a027ec/
```

- 3 ユーザー操作オブジェクトの URL で、現在実行中のワークフローを実行するため、以下の **POST** 要求を作成します。

```
POST https://localhost:
8281/vco/api/workflows/CF808080808080808080808080E6808080013086668236014a0614d16e
1/executions/8880808080808080808080808080578080800132145338690643f66a027ec/interacti
on/
```

要求本文内で、以下のように入力パラメータの値を指定します。

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

入力パラメータの検証後にユーザー操作に応答する

ユーザー操作のプレゼンテーションにより、ワークフローの入力パラメータに渡すことができる値に対する制約が定義される場合があります。ユーザー操作に応答する際に、そのユーザー操作のプレゼンテーション内で定義されている制約に対して、入力パラメータに渡す値を検証することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 有効なユーザー操作オブジェクトが存在する URL で **GET** 要求を行いすべてのユーザー操作オブジェクトのリストを取得するか、または待機中のユーザー操作オブジェクトのみをフィルタリングして取得します。

URL	説明
<code>https://<orchestrator_host>:<port>/vco/api/catalog/System/UserInteraction</code>	Orchestrator の有効なユーザー操作オブジェクトが存在する URL
<code>https://<orchestrator_host>:<port>/vco/api/catalog/System/UserInteraction?status=0</code>	待機中のユーザー操作オブジェクトのみをフィルタリングする URL

有効なユーザー操作オブジェクトのリストが取得されます。待機中のユーザー操作には、値が **waiting** で **state** という名前の属性が設定されています。

- 2 応答したい待機中のユーザー操作のインベントリ アイテムが存在する URL で、以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
<port>}/vco/api/catalog/System/UserInteraction/{<userInteractionID>}/
```

応答本文に、ユーザー操作インスタンスに対するリンクが表示されます。このユーザー操作インスタンスには、特定のワークフローの実行が関連付けられています。

- 3 ユーザー操作インスタンスの URL で、以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/interaction/
```

応答本文に、ユーザー操作のプレゼンテーションに対するダウン リンクが表示されます。

- 4 ユーザー操作のプレゼンテーションの URL で、以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/interaction/presentation/
```

応答本文に、ユーザー操作のプレゼンテーションの定義が表示されます。

- 5 プレゼンテーションの定義で、入力パラメータに渡すことができる値の制限を確認します。
- 6 プレゼンテーションのインスタンスが存在する URL で以下の **POST** 要求を作成して、ユーザー操作のプレゼンテーションを実行します。

```
POST https://{<orchestrator_host>}:
{<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/interaction/presentation/instances/
```

- 7 要求本文内の **execution-context** 要素で、入力パラメータの値を指定します。

応答本文に、ユーザー操作のプレゼンテーションのインスタンスが表示されます。入力パラメータに渡された値が正しい場合、**execution** 要素に **valid="true"** という属性が表示されます。また、**output-parameters** 要素には、入力パラメータの正しい値が表示されます。これらの値を使用して、ユーザー操作に応答することができます。

- 8 ユーザー操作インスタンスが存在する URL で以下の **POST** 要求を作成して、ユーザー操作に応答します。

```
POST https://{<orchestrator_host>}:
{<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/interaction/
```

- 9 要求本文内で、入力パラメータの値を持つ **execution-context** コンテキストを渡します。

ユーザー操作プレゼンテーションの URL で作成した **POST** 要求の場合と同じ要求本文を使用することができます。

最後の要求が正常に実行されると、ステータス コード 204 と空の応答本文を受信します。

例: 入力パラメータを検証することにより、対話式の「Hello World」ワークフローのユーザー操作に応答します。

ユーザー操作のプレゼンテーションで定義されている制約に対して入力パラメータの値を検証することにより、対話式の「Hello World」ワークフローのユーザー操作に応答することができます。

- 1 カタログ サービスのユーザー操作オブジェクトのエンドポイントで以下の **GET** 要求を作成して、ワークフローの待機中のユーザー操作を検索します。

```
GET https://localhost:8281/vco/api/catalog/System/UserInteraction?status=0
```

- 2 対話式の「Hello World」ワークフローのユーザー操作インベントリ オブジェクトを探し、そのオブジェクトの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080805A8080800132
145338690643f66a027ec/
```

- 3 ユーザー操作インスタンスの URL で、以下の **GET** 要求を作成します。

```
GET https://localhost:
8281/vco/api/catalog/System/UserInteraction/88808080808080808080808080805A8080800132
145338690643f66a027ec/interaction/
```

- 4 ユーザー操作プレゼンテーションの URL で、以下の **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/catalog/System/UserInteraction/888080808080808080808080808080805A8080800132  
145338690643f66a027ec/interaction/presentation/
```

このプレゼンテーションにより、入力パラメータが必須パラメータとして定義されます。このプレゼンテーションには、渡すことのできる文字列の長さに関する制約が含まれています。

- 5 ユーザー操作プレゼンテーションのインスタンスが存在する URL で、以下の **POST** 要求を作成します。

```
POST https://localhost:8281/vco/api/catalog/System/UserInteraction/8880808080808080808080805A8080800132145338690643f66a027ec/interaction/presentation/instances/
```

要求本文内で、以下のように入力パラメータの値を指定します。

```
<execution-context xmlns="http://www.vmware.com/vco">
  <parameters>
    <parameter name="name" type="string">
      <string>John Smith</string>
    </parameter>
  </parameters>
</execution-context>
```

応答本文内の **execution** 要素に、**valid="true"** という属性が設定されます。この属性は、ユーザー操作プレゼンテーション内の制約に対して、入力パラメータの値が正しいことを示しています。正しい値は、以下の **output-parameters** 要素で確認することができます。

```
.....  
<output-parameters>  
  <parameter name="name" type="string">  
    <string>John Smith</string>  
  </parameter>  
</output-parameters>  
.....
```

- 手順 5 の **POST** 要求の場合と同じ要求本文を渡して、ユーザー操作インスタンスの URL で以下の **POST** 要求を作成します。

```
POST https://localhost:  
8281/vco/api/catalog/System/UserInteraction/8880808080808080808080808080805A8080800132  
145338690643f66a027ec/interaction/
```

ワークフローの実行のキャンセル

Orchestrator REST API を使用して、ワークフローの実行をキャンセルすることができます。

手順

- 1 ワークフローの定義の URL で以下の **GET** 要求を作成して、ワークフローの定義を取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

ワークフローの定義には、そのワークフローの有効な実行オブジェクトに対するリンクが含まれています。

- 2 ワークフローの有効な実行オブジェクトが存在する URL で以下の **GET** 要求を作成して、実行中の有効なワークフローを取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/executions/
```

- 3 実行中の有効なワークフローのリストで、キャンセルしたいワークフローを選択し、そのワークフローの URL で以下の **DELETE** 要求を作成します。

```
DELETE https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/executions/{<executionID>}/state
```

ワークフロー操作の取得

Orchestrator REST API を使用して、ワークフローに関連するすべてのユーザー操作のリストを取得することができます。

手順

- 1 ワークフローの定義の URL で以下の **GET** 要求を作成して、ワークフローの定義を取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

- 2 ワークフロー操作の URL に対して以下の **GET** 要求を作成して、ワークフロー操作のリストを取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/interactions/
```

GET 要求が正常に処理されると、ステータス コード 200 とともに、ワークフローで使用可能なすべてのユーザー操作のリストを受信します。

ワークフローのスキーマへのアクセス

Orchestrator REST API を使用して、ワークフローのスキーマ イメージにアクセスすることができます。

手順

- 1 ワークフローの定義の URL で以下の **GET** 要求を作成して、ワークフローの定義を取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

- 2 ワークフローのスキーマの URL に対して以下の **GET** 要求を作成して、ワークフローのスキーマを取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/schema/
```

GET 要求が正常に処理されると、ステータス コード 200 と、ワークフローのスキーマを表すイメージのバイナリデータを受信します。その後、応答コンテンツ タイプが正しいメディア タイプに設定されます (例: **Content-Type: image/png**)。

タスクの操作

Orchestrator REST API のタスク サービスを使用すると、Orchestrator の管理タスクに関連するすべての操作を実行できます。実行可能な操作には、ワークフローのスケジュール設定タスクの作成、既存のタスクのプロパティの変更、タスクの削除などがあります。

Orchestrator がサポートするスケジュール設定タスクの最大数は 50 です。

タスクの作成

Orchestrator REST API を使用して、ワークフローのスケジュールを設定するためのタスクを作成することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 ワークフローの URL で以下の **GET** 要求を作成して、タスクの作成対象となるワークフローの定義を取得します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

ワークフローの定義では、そのワークフローの名前と ID だけでなく、入力パラメータも確認することができます。

- 2 ワークフローに対して新しいタスクを作成するには、タスク サービスの URL で以下の **POST** 要求を作成します。

```
POST https://{<orchestrator_host>}:{<port>}/vco/api/tasks/
```

- 3 要求本文内の **task** 要素で、新しいタスクのパラメータを指定します。

要求が正常に処理されると、API からステータス コード 202 と空の応答本文が返されます。

手順

- 1 変更するタスクの URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/tasks/{<task ID>}/
```

- 2 要求の応答本文でタスクのプロパティを確認します。
- 3 タスクを変更するには、要求本文内の **task-data** 要素でタスクの新しいプロパティを指定して、タスクの URL で以下の POST 要求を作成します。

POST 要求が正常に処理されると、API が応答本文中にステータス コード 200 と更新されたタスクを返します。

例: 「Hello の送信」 サンプル タスクの更新

タスクの開始日と終了日を更新できます。[「タスクの作成」](#) で紹介したサンプル タスクを変更できます。要求本文に新しい開始日と終了日を指定して、タスクの URL で以下の POST 要求を作成します。

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<task-data xmlns="http://www.vmware.com/vco">
  <recurrence-start-date>2012-02-01T14:00:00+02:00</recurrence-start-date>
  <recurrence-end-date>2012-02-05T14:00:00+02:00</recurrence-end-date>
</task-data>
```

タスクの状態の確認

現在使用可能なタスクの状態を確認することも、特定のタスクのすべての実行インスタンスの状態を確認することもできます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 現在使用可能なすべてのタスクの状態を確認するには、タスク サービスの URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/tasks/
```

応答本文に、Orchestrator で現在使用可能なタスクの定義が表示されます。各タスクの状態は、**state** という名前の **attribute** 要素で確認することができます。この要素の有効な値は、**finished**、**pending**、**running** などです。

- 特定のタスクのすべての実行インスタンスの状態を確認するには、そのタスクの実行インスタンスが存在する URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/tasks/{<taskID>}/executions/
```

応答本文に、該当するタスクの使用可能な実行インスタンスのリストが表示されます。各実行インスタンスの状態は、タスク実行オブジェクトの **state** 要素で確認することができます。

Orchestrator インベントリでのオブジェクトの検索

カタログ サービスまたはインベントリ サービスを使用して、Orchestrator インベントリ内の任意のオブジェクトを検索することができます。HTTP 要求を作成する URL の最後にフィルタ パラメータを指定することにより、オブジェクトの特定のサブセットだけに絞り込んでアクセスすることができます。

カタログ サービスを使用して、特定のタイプのオブジェクトを Orchestrator インベントリで検索したり、オブジェクトのタイプと ID を指定して特定のオブジェクトを取得したりすることができます。たとえば、タイプが **workflow** または **action** のオブジェクトをすべて取得することも、特定のワークフローやアクションだけを取得することもできます。

インベントリ サービスを使用すると、親子関係別に Orchestrator インベントリを参照することができます。また、インベントリ サービスにより、Orchestrator インベントリ内の特定の場所でのみ使用可能なオブジェクトにアクセスすることもできます。たとえば、Orchestrator インベントリ内のワークフローの場所 (**Library/vCenter/Datacenter**) を参照することにより、データセンター管理機能用のすべてのワークフローを取得することができます。

HTTP 要求を作成する際に、Orchestrator REST API のすべてのサービスで、URL の最後にフィルタ パラメータを追加することができます。フィルタ パラメータを使用すると、特定の URL の応答本文に表示される実行結果の範囲を絞り込むことができます。

タイプと ID によるオブジェクトの検索

REST API のカタログ サービスでタイプと ID を使用して、Orchestrator 内のオブジェクトを検索することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 カatalog サービスの URL で以下の GET 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/catalog/
```

要求に対する応答本文に、Orchestrator のインベントリを公開するプラグインのカatalog エントリ ポイントに対するダウンリンクと、Orchestrator 内のシステム オブジェクトに対するダウンリンクが表示されます。

- `https://{<orchestrator_host>}:{<port>}/vco/api/catalog/{<plug-in namespace>}/`
- `https://{<orchestrator_host>}:{<port>}/vco/api/catalog/System/`

- 2 プラグインによって公開されるオブジェクトまたは Orchestrator のシステム オブジェクトにアクセスするには、そのプラグインのカatalog エントリ ポイントの URL または Orchestrator のシステム オブジェクトが存在する URL で **GET** 要求を作成します。

要求に対する応答本文に、公開されたオブジェクト タイプのリンクが表示されます。

- 3 アクセスするオブジェクトタイプの URL で以下の **GET** 要求を作成します。

GET https://{<orchestrator_host>}:{<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/

- #### 4 検索するオブジェクトの URL で以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:
      {<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/{<objectID>}/
```

例: 「Hello の送信」 ワークフローの検索

カタログ サービスを使用して、サンプルの「Hello の送信」ワークフローを検索することができます。

- 1 カタログサービスの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/catalog/
```

- 2 Orchestrator のすべてのシステム オブジェクトが存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/catalog/System/
```

- 3 すべてのワークフローが存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/catalog/Workflow/
```

- 4 「Hello の送信」 ワークフローの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:  
8281/vco/api/catalog/Workflow/CF808080808080808080808080DA808080013086668236014a06  
14d16e1/
```

関係を使用したオブジェクトの検索

Orchestrator REST のインベントリ サービスを使用して、Orchestrator とプラグインのインベントリを階層として参照することができます。

開始する前に

Orchestrator にサンプルのワークフロー パッケージがインポート済みであることを確認します。パッケージは、Orchestrator のドキュメント ページからダウンロードできる Orchestrator サンプル アプリケーションの ZIP ファイルに含まれています。

手順

- 1 インベントリ サービスの URL で以下の **GET** 要求を作成します。

```
GET https://{<orchestrator_host>}:{<port>}/vco/api/inventory/
```

応答本文に、インストールされているプラグインの登録済みインベントリに対するダウン リンクが表示され、**System** というテキストの下に、Orchestrator 内のシステム オブジェクトに対するダウン リンクが表示されます。

- 2 アクセスしたいインベントリのダウン リンクで **GET** 要求を作成します。
- 3 目的のオブジェクトが見つかるまで、インベントリ内のアイテムのアップ リンクとダウン リンクで **GET** 要求を作成します。

例: 「Hello の送信」 ワークフローの検索

Orchestrator インベントリを参照して、「Hello の送信」 ワークフローを検索することができます。

- 1 インベントリ サービスの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/inventory/
```

- 2 Orchestrator のシステム オブジェクトが存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/inventory/System/
```

- 3 Orchestrator のすべてのワークフローが存在する URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/
```

- 4 サンプルのワークフロー カテゴリの URL で以下の **GET** 要求を作成します。

```
GET https://localhost:8281/vco/api/inventory/System/Workflows/Samples/
```

- 5 「Hello World」 ワークフロー カテゴリのダウン リンクを使用して、「Hello の送信」 ワークフローを検索します。

フィルタの適用

Orchestrator REST API の各サービスは、API から HTTP 要求に対して返されるオブジェクトの数を絞り込むための追加の URL パラメータをサポートしています。

REST API 経由でアクセス可能なリソースに対する各 URL について、異なるクエリ パラメータを使用することができます。URL で使用できるクエリ パラメータについては、vRealize Orchestrator REST API のリファレンス ドキュメントを参照してください。

手順

- ◆ 要求の実行結果を特定の URL で絞り込むには、その URL の最後にフィルタを指定します。

<URL>?<filter_1>& <filter_2>&<filter_3>&...&<filter_N> すべてのフィルタで、関連する URL に対して有効なクエリ パラメータを指定します。各 URL に対する有効なクエリ パラメータについては、Orchestrator REST API のリファレンス ドキュメントを参照してください。

例: ワークフローのフィルタ

名前に特定の単語が含まれているワークフローを検索する場合は、要求内で以下のフィルタをカタログ サービスに適用します。以下の例では、名前に「datastore」という単語が含まれているワークフローが検索されます。

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?conditions=name~datastore
```

検索結果として返されるワークフローの数を絞り込むには、追加のフィルタを要求内で指定します。以下の例では、検索結果として返される件数を最大 5 件に絞り込んでいます。

```
GET https://localhost:8281/vco/api/catalog/System/Workflow?
conditions=name~datastore&maxResult=5
```

Orchestrator オブジェクトのインポートとエクスポート

Orchestrator REST API には、ワークフロー、アクション、パッケージ、リソース、構成要素をインポートおよびエクスポートするための Web サービスが付属しています。

ワークフローのインポート

Orchestrator REST API を使用して、ワークフローをインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、ワークフローのプロパティを定義するカスタム コードを使用できます。

開始する前に

ワークフローのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、インポートするワークフローのプロパティを定義するための要求ヘッダーを追加します。
- 2 ワークフロー オブジェクトの URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/workflows/
```

POST 要求が正常に処理されると、ステータス コード 202 を受信します。

ワークフローのエクスポート

Orchestrator REST API を使用して、ワークフローをエクスポートすることができます。エクスポートしたワークフローは、ファイルとしてダウンロードすることができます。

手順

1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。

- [名前]: **accept**
- [値]: **application/zip**

2 エクスポートするワークフローの URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。ワークフローのバイナリ コンテンツは、**<workflow_name>.workflow** というデフォルトのファイル名を持つ添付ファイルとして使用することができます。このファイルは、REST クライアント アプリケーションを使用して保存することができます。

アクションのインポート

Orchestrator REST API を使用して、アクションをインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、アクションのプロパティを定義するカスタム コードを使用できます。

開始する前に

アクションのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、インポートするアクションのプロパティを定義するための要求ヘッダーを追加します。
- 2 アクション オブジェクトの URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/actions/
```

POST 要求が正常に処理されると、ステータス コード 202 を受信します。

アクションのエクスポート

Orchestrator REST API を使用して、アクションをエクスポートすることができます。エクスポートしたアクションは、ファイルとしてダウンロードすることができます。

手順

1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。

- [名前]: **accept**
- [値]: **application/zip**

2 エクスポートするアクションの URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/{<actionID>}/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。アクションのバイナリ コンテンツは、**<action_name>.action** というデフォルトのファイル名を持つ添付ファイルとして使用することができます。このファイルは、REST クライアント アプリケーションを使用して保存することができます。

パッケージのインポート

Orchestrator REST API を使用して、パッケージをインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、パッケージのプロパティを定義するカスタム コードを使用できます。

重複した名前を使用して Orchestrator のパッケージをインポートした場合、デフォルトでは、既存のパッケージは上書きされません。既存のパッケージを上書きするかどうかを指定するには、要求内でパラメータを指定します。

デフォルトでは、Orchestrator のパッケージは、構成要素の属性値とともにインポートされます。属性値を除いてパッケージをインポートするには、要求内でパラメータを指定します。

Orchestrator パッケージに含まれているタグは、デフォルトではインポートされますが、Orchestrator サーバに同じタグがすでに存在している場合は、既存のタグの値が保持されます。既存のタグの値を保持するかどうかを指定するには、要求内でパラメータを指定します。

開始する前に

パッケージのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

1 REST クライアント アプリケーションで、インポートするパッケージのプロパティを定義するための要求ヘッダーを追加します。

- 2 パッケージ オブジェクトの URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/
```

- 3 (オプション) パッケージをインポートし、同じ名前の既存のパッケージを上書きするには、以下のように **POST** 要求内で **overwrite** パラメータを指定します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/?overwrite=true
```

- 4 (オプション) 構成要素の属性値を除いてパッケージをインポートするには、以下のように **POST** 要求内で **importConfigurationAttributeValues** パラメータを指定します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/?
importConfigurationAttributeValues=false
```

- 5 (オプション) 含まれているタグを除いてパッケージをインポートするには、以下のように **POST** 要求内で **tagImportMode** パラメータを指定します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/?tagImportMode=DoNotImport
```

- 6 (オプション) 含まれているタグを除いてパッケージをインポートし、既存のタグ値を上書きするには、以下のように **POST** 要求内で **tagImportMode** パラメータを指定します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/?
tagImportMode=ImportAndOverwriteExistingValue
```

POST 要求が正常に処理されると、ステータス コード 202 を受信します。

パッケージのエクスポート

Orchestrator REST API を使用して、パッケージをエクスポートすることができます。エクスポートしたパッケージは、ファイルとしてダウンロードすることができます。

デフォルトでは、Orchestrator のパッケージは、構成要素の属性値とグローバル タグとともにエクスポートされます。要求内でパラメータを使用することにより、属性値やグローバル タグを除いてパッケージをエクスポートすることができます。また、ダウンロードするパッケージ ファイルのカスタム名を指定することもできます。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。

- [名前]: **accept**
- [値]: **application/zip**

- 2 エクスポートするパッケージの URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/
```

- 3 (オプション) エクスポートするパッケージのカスタム名を設定するには、以下のように **GET** 要求内で **packageName** パラメータを指定します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/?
packageName={<custom_name>}
```

- 4 (オプション) 構成要素の属性値を除いてパッケージをダウンロードするには、以下のように **GET** 要求内で **exportConfigurationAttributeValues** パラメータを指定します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/?
exportConfigurationAttributeValues=false
```

- 5 (オプション) グローバル タグを除いてパッケージをエクスポートするには、以下のように **GET** 要求内で **exportGlobalTags** パラメータを指定します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/?
exportGlobalTags=false
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。パッケージのバイナリ コンテンツは、**<package_name>.package** というデフォルトのファイル名を持つ添付ファイルとして使用することができます。このファイルは、REST クライアント アプリケーションを使用して保存することができます。

リソースのインポート

Orchestrator REST API を使用して、リソースをインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、リソースのプロパティを定義するカスタム コードを使用できます。

開始する前に

リソースのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、インポートするリソースのプロパティを定義するための要求ヘッダーを追加します。
- 2 リソース オブジェクトの URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/resources/
```

POST 要求が正常に処理されると、ステータス コード 202 を受信します。

リソースのエクスポート

Orchestrator REST API を使用して、リソースをエクスポートすることができます。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。

- [名前]: **accept**
- [値]: **application/octet-stream**

- 2 エクスポートするリソースの URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/{<resourceID>}/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、応答本文内でリソースのコンテンツが使用可能になります。

構成要素のインポート

Orchestrator REST API を使用して、構成要素をインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、構成要素のプロパティを定義するカスタム コードを使用できます。

開始する前に

構成要素のバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、インポートする構成要素のプロパティを定義するための要求ヘッダーを追加します。
- 2 構成要素 オブジェクトの URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/configurations/
```

POST 要求が正常に処理されると、ステータス コード 202 を受信します。

構成要素のエクスポート

Orchestrator REST API を使用して、構成要素をエクスポートすることができます。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。

- [名前]: **accept**
- [値]: **application/vcoobject+xml**

- 2 エクスポートする構成要素の URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:  
{<port>}/vco/api/configurations/{<configuration_elementID>}/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、応答本文内で構成要素のコンテンツが使用可能になります。

Orchestrator オブジェクトの削除

Orchestrator REST API には、ワークフロー、アクション、パッケージ、リソース、構成要素を削除するための Web サービスが付属しています。

ワークフローの削除

Orchestrator REST API を使用して、ワークフローを削除することができます。

手順

- 1 次のように、**GET** 要求を発行して、返されたワークフローのリストからワークフローの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/
```

- 2 ワークフローの URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/
```

DELETE 要求が正常に処理されると、ステータス コード 200 と空の応答本文が返されます。

アクションの削除

Orchestrator REST API を使用して、アクションを削除することができます。

手順

- 1 次のように、**GET** 要求を発行し、返されたアクションのリストからアクションの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/
```

- 2 アクションの URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/actions/{<actionID>}/
```

DELETE 要求が正常に処理されると、ステータス コード 200 と空の応答本文が返されます。

パッケージの削除

Orchestrator REST API を使用して、パッケージを削除することができます。

パッケージを削除しても、そのパッケージの要素は削除されません。パッケージのコンテンツも削除したい場合は、オプションのパラメータを指定する必要があります。

手順

- 1 次のように、**GET** 要求を発行して、返されたパッケージのリストからパッケージ名を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/
```

- 2 パッケージの URL で **DELETE** 要求を作成します。パッケージの要素も削除したい場合は、以下のように、DELETE 要求の最後にオプションのパラメータを指定します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/?
option={<parameter>}
```

パラメータ	説明
deletePackage	パッケージだけが削除され、パッケージのコンテンツは残ったままになります。
deletePackageWithContent	パッケージとそのすべてのコンテンツが削除されます。削除されたパッケージの要素が別のパッケージで共有されている場合、そのパッケージからも共有要素が削除されます。
deletePackageKeepingShared	パッケージが削除され、他のパッケージと共有されていないコンテンツが削除されます。他のパッケージと共有されている要素は削除されません。

オプションのパラメータを指定しなかった場合は、デフォルト パラメータの **deletePackage** が使用されます。

DELETE 要求が正常に処理されると、ステータス コード 200 と空の応答本文が返されます。

リソースの削除

Orchestrator REST API を使用して、リソースを削除することができます。

手順

- 1 次のように、**GET** 要求を発行して、返されたリソースのリストからリソースの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/
```

- 2 リソースの URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/resources/{<resourceID>}/
```

DELETE 要求が正常に処理されると、ステータス コード 200 と空の応答本文が返されます。

構成要素の削除

Orchestrator REST API を使用して、構成要素を削除することができます。

手順

- 1 次のように、**GET** 要求を発行して、返された構成要素のリストから構成要素の ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/configurations/
```

- 2 構成要素の URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:  
{<port>}/vco/api/configurations/{<configuration_elementID>}/
```

DELETE 要求が正常に処理されると、ステータス コード 200 と空の応答本文が返されます。

Orchestrator オブジェクトに対する権限の設定

REST API を使用して Orchestrator オブジェクトのカスタム権限を設定できます。権限を設定するには、オブジェクトの権限の URL で **POST** 要求を作成し、要求本文で権限を定義します。

Orchestrator REST API を使用してオブジェクトの権限に関する情報を取得することも、既存の権限を削除することもできます。

REST API 権限

Orchestrator REST API を使用して権限を設定する場合、文字セットを使用して権限を定義する必要があります。

要素の権限を設定するには、**POST** 要求の要求本文の **<rights>** タグに一連の文字を指定します。

Orchestrator REST API を使用して権限を設定する際に使用できる文字には特定の意味があります。

表 2-1. Orchestrator REST API 権限の文字セット

文字	説明
r	表示権限を付与します。
x	実行権限を付与します。
i	検査権限を付与します。
c	編集権限を付与します。
a	管理者権限を付与します。

例: 権限を設定するための構文

Orchestrator 要素の権限の URL で、POST 要求の要求本文に以下の例の構文を使用できます。

```
<permissions xmlns="http://www.vmware.com/vco">
  <permission>
    <principal>cn=vcousers,ou=vco,dc=appliance</principal>
    <rights>ric</rights>
  </permission>
</permissions>
```

要求本文の **<rights>** タグに **ric** 権限を設定すると、**vcousers** ユーザー グループのメンバーは Orchestrator 要素を表示、検査、編集できるようになります。

ワークフローの権限の取得

Orchestrator REST API を使用して、ワークフローの権限に関する情報を取得することができます。

手順

- 1 次のように、GET 要求を発行して、返されたワークフローのリストからワークフローの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/
```

- 2 ワークフローの権限の URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/permissions/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、ワークフローの権限に関する情報が応答本文で使用可能になります。

ワークフローの権限の削除

Orchestrator REST API を使用して、ワークフローの権限を削除することができます。ワークフローの既存の権限を削除してから、新しい権限を設定することができます。

手順

- 1 次のように、GET 要求を発行して、返されたワークフローのリストからワークフローの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/
```

- 2 ワークフローの権限の URL で以下の DELETE 要求を作成します。

```
DELETE http://{<orchestrator_host>}:
{<port>}/vco/api/workflows/{<workflowID>}/permissions/
```

DELETE 要求が正常に処理されると、ステータス コード 204 と空の応答本文が返されます。

ワークフローの権限の設定

Orchestrator REST API を使用して、ワークフローの権限を設定することができます。

開始する前に

設定可能な権限のタイプと、要求本文で使用可能な構文を確認します。[「REST API 権限」](#)を参照してください。

手順

- 1 次のように、**GET** 要求を発行して、返されたワークフローのリストからワークフローの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/workflows/
```

- 2 REST クライアント アプリケーションで、権限を設定するワークフローのプロパティを定義するための要求ヘッダーを追加します。
- 3 要求本文で、設定する権限を指定します。
- 4 ワークフローの権限の URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/workflows/{<workflowID>}/permissions/
```

POST 要求が正常に処理されると、ステータス コード 201 を受信します。また、ワークフローの権限に関する情報が応答本文で使用可能になります。

アクションの権限の取得

Orchestrator REST API を使用して、アクションの権限に関する情報を取得することができます。

手順

- 1 次のように、**GET** 要求を発行し、返されたアクションのリストからアクションの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/
```

- 2 アクションの権限の URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/{<actionID>}/permissions/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、アクションの権限に関する情報が応答本文で使用可能になります。

アクションの権限の削除

Orchestrator REST API を使用して、アクションの権限を削除することができます。アクションの既存の権限を削除してから、新しい権限を設定することができます。

手順

- 1 次のように、**GET** 要求を発行し、返されたアクションのリストからアクションの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/
```

- 2 アクションの権限の URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/actions/{<actionID>}/permissions/
```

DELETE 要求が正常に処理されると、ステータス コード 204 と空の応答本文が返されます。

アクションの権限の設定

Orchestrator REST API を使用して、アクションの権限を設定することができます。

開始する前に

設定可能な権限のタイプと、要求本文で使用可能な構文を確認します。[「REST API 権限」](#)を参照してください。

手順

- 1 次のように、**GET** 要求を発行し、返されたアクションのリストからアクションの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/actions/
```

- 2 REST クライアント アプリケーションで、権限を設定するアクションのプロパティを定義するための要求ヘッダーを追加します。
- 3 要求本文で、設定する権限を指定します。
- 4 アクションの権限の URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/actions/{<actionID>}/permissions/
```

POST 要求が正常に処理されると、ステータス コード 201 を受信します。また、アクションの権限に関する情報が応答本文で使用可能になります。

パッケージの権限の取得

Orchestrator REST API を使用して、パッケージの権限に関する情報を取得することができます。

手順

- 1 次のように、**GET** 要求を発行して、返されたパッケージのリストからパッケージ名を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/
```

- 2 パッケージの権限の URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/permissions/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、パッケージの権限に関する情報が応答本文で使用可能になります。

パッケージの権限の削除

Orchestrator REST API を使用して、パッケージの権限を削除することができます。パッケージの既存の権限を削除してから、新しい権限を設定することができます。

手順

- 1 次のように、**GET** 要求を発行して、返されたパッケージのリストからパッケージ名を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/
```

- 2 パッケージの権限の URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/permissions/
```

DELETE 要求が正常に処理されると、ステータス コード 204 と空の応答本文が返されます。

パッケージの権限の設定

Orchestrator REST API を使用して、パッケージの権限を設定することができます。

開始する前に

設定可能な権限のタイプと、要求本文で使用可能な構文を確認します。[「REST API 権限」](#)を参照してください。

手順

- 1 次のように、**GET** 要求を発行して、返されたパッケージのリストからパッケージ名を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/packages/
```

- 2 REST クライアント アプリケーションで、権限を設定するパッケージのプロパティを定義するための要求ヘッダーを追加します。
- 3 要求本文で、設定する権限を指定します。
- 4 パッケージの権限の URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/packages/{<package_name>}/permissions/
```

POST 要求が正常に処理されると、ステータス コード 201 を受信します。また、パッケージの権限に関する情報が応答本文で使用可能になります。

リソースの権限の取得

Orchestrator REST API を使用して、リソースの権限に関する情報を取得することができます。

手順

- 1 次のように、GET 要求を発行して、返されたリソースのリストからリソースの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/
```

- 2 リソースの権限の URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/{<resourceID>}/permissions/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、リソースの権限に関する情報が応答本文で使用可能になります。

リソースの権限の削除

Orchestrator REST API を使用して、リソースの権限を削除することができます。リソースの既存の権限を削除してから、新しい権限を設定することができます。

手順

- 1 次のように、GET 要求を発行して、返されたリソースのリストからリソースの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/
```

- 2 リソースの権限の URL で以下の DELETE 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/resources/{<resourceID>}/permissions/
```

DELETE 要求が正常に処理されると、ステータス コード 204 と空の応答本文が返されます。

リソースの権限の設定

Orchestrator REST API を使用して、リソースの権限を設定することができます。

開始する前に

設定可能な権限のタイプと、要求本文で使用可能な構文を確認します。[\[REST API 権限\]](#) を参照してください。

手順

- 1 次のように、**GET** 要求を発行して、返されたリソースのリストからリソースの ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/resources/
```

- 2 REST クライアント アプリケーションで、権限を設定するリソースのプロパティを定義するための要求ヘッダーを追加します。
- 3 要求本文で、設定する権限を指定します。
- 4 リソースの権限の URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/resources/{<resourceID>}/permissions/
```

POST 要求が正常に処理されると、ステータス コード 201 を受信します。また、リソースの権限に関する情報が応答本文で使用可能になります。

構成要素の権限の取得

Orchestrator REST API を使用して、構成要素の権限に関する情報を取得することができます。

手順

- 1 次のように、**GET** 要求を発行して、返された構成要素のリストから構成要素の ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/configurations/
```

- 2 構成要素の権限の URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/configurations/{<configuration_elementID>}/permissions/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、構成要素の権限に関する情報が応答本文で使用可能になります。

構成要素の権限の削除

Orchestrator REST API を使用して、構成要素の権限を削除することができます。構成要素の既存の権限を削除してから、新しい権限を構成することができます。

手順

- 1 次のように、**GET** 要求を発行して、返された構成要素のリストから構成要素の ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/configurations/
```

- 2 構成要素の権限の URL で以下の **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:  
{<port>}/vco/api/configurations/{<configuration_elementID>}/permissions/
```

DELETE 要求が正常に処理されると、ステータス コード 204 と空の応答本文が返されます。

構成要素の権限の構成

Orchestrator REST API を使用して、構成要素の権限を構成することができます。

開始する前に

設定可能な権限のタイプと、要求本文で使用可能な構文を確認します。[「REST API 権限」](#)を参照してください。

手順

- 1 次のように、**GET** 要求を発行して、返された構成要素のリストから構成要素の ID を取得します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/configurations/
```

- 2 REST クライアント アプリケーションで、権限を構成する構成要素のプロパティを定義するための要求ヘッダーを追加します。
- 3 要求本文で、設定する権限を指定します。
- 4 構成要素の権限の URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:  
{<port>}/vco/api/configurations/{<configuration_elementID>}/permissions/
```

POST 要求が正常に処理されると、ステータス コード 201 を受信します。また、構成要素の権限に関する情報が応答本文で使用可能になります。

プラグインを使用した操作の実行

Orchestrator REST API には、プラグインを使用した各種操作の実行に使用できる Web サービスが付属しています。

プラグインに関する情報の取得

Orchestrator REST API を使用して、インストールされているすべてのプラグインに関するメタデータ情報を取得することができます。

手順

- 1 REST クライアント アプリケーションで、プラグインのプロパティを定義するための要求ヘッダーを追加します。

- 2 プラグイン オブジェクトの URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/plugins/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。

プラグインのインポート

Orchestrator REST API を使用して、プラグインをインポートすることができます。

REST クライアント アプリケーションのライブラリに応じて、プラグインのプロパティを定義するカスタム コードを使用できます。

注意 同じ名前のプラグインがすでにインストールされている場合は、プラグインをインポートできません。

開始する前に

プラグインのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、インポートするプラグインのプロパティを定義するための要求ヘッダーを追加します。
- 2 プラグイン オブジェクトの URL で以下の **POST** 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/plugins/
```

POST 要求が正常に処理されると、ステータス コード 200 を受信します。

プラグインのエクスポート

Orchestrator REST API を使用して、プラグインをエクスポートすることができます。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。
 - [名前]: **accept**
 - [値]: **application/dar**
- 2 エクスポートするプラグインの URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/plugins/{<plug-in_name>}/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。また、応答本文内でプラグインのコンテンツが使用可能になります。

プラグインの有効と無効の切り替え

Orchestrator REST API を使用して、プラグインの有効と無効を切り替えることができます。

プラグインの URL で **PUT** 要求を作成することにより、プラグインの有効と無効を切り替えることができます。

Orchestrator プラグインに関する情報を取得すると、プラグインの現在の状態を確認することができます。[「プラグインに関する情報の取得」](#)を参照してください。

開始する前に

プラグインのバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、有効と無効を切り替えるプラグインのプロパティを定義するための要求ヘッダーを追加します。
- 2 有効と無効を切り替えるプラグインの URL で以下の **PUT** 要求を作成します。

```
PUT http://{<orchestrator_host>}:{<port>}/vco/api/plugins/{<plug-in_name>}/state/
```

PUT 要求が正常に処理されると、ステータス コード 200 を受信します。

サーバ構成操作の実行

Orchestrator REST API には、Orchestrator サーバ構成に関連する各種操作の実行に使用可能な Web サービスが付属しています。

Orchestrator サーバ構成に関する情報の取得

Orchestrator REST API を使用して、Orchestrator サーバ構成に関する情報を取得することができます。

手順

- 1 REST クライアント アプリケーションで、情報を取得するサーバのプロパティを定義するための要求ヘッダーを追加します。
- 2 プラグイン オブジェクトの URL で以下の **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/server-configuration/
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。

Orchestrator サーバ構成のインポート

Orchestrator REST API を使用して、保存されている構成をインポートすることができます。

開始する前に

サーバ構成のバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。
 - [名前]: **content-type**
 - [値]: **multipart/form-data**
- 2 サーバ構成の URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/server-configuration/
```

POST 要求が正常に処理されると、ステータス コード 200 を受信します。

Orchestrator サーバ構成のエクスポート

Orchestrator REST API を使用して、サーバ構成をエクスポートすることができます。

開始する前に

サーバ構成のバイナリ コンテンツは、マルチパート コンテンツとして使用可能になっている必要があります。詳細については、RFC 2387 を参照してください。

手順

- 1 REST クライアント アプリケーションで、以下の値を持つ要求ヘッダーを追加します。
 - [名前]: **content-type**
 - [値]: **multipart/form-data**
- 2 以下の値を指定して、別の要求ヘッダーを追加します。
 - [名前]: **accept**
 - [値]: ***/***
- 3 サーバ構成の URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:{<port>}/vco/api/server-configuration/
```

POST 要求が正常に処理されると、ステータス コード 200 を受信します。

タグ付け操作の実行

Orchestrator REST API には、Orchestrator でタグを使用してオブジェクトを検索しやすくする各種操作を実行するための Web サービスが付属しています。

オブジェクトはタグ付けして検索しやすくすることができます。タグはスペース文字を含まない 3 ～ 64 文字の長さの文字列です。

グローバル タグとプライベート タグを追加できます。グローバル タグはすべての Orchestrator ユーザーに表示され、プライベート タグは作成者にのみ表示されます。グローバル タグは管理者権限を持つユーザーのみが作成、削除できます。

オブジェクトのタグ付け

Orchestrator REST API を使用して、オブジェクトにタグを割り当てることができます。

プライベート タグとグローバル タグの両方を作成できます。要求本文でタグがプライベートかグローバルかを指定します。

注意 グローバル タグを作成するには、管理者権限を持つユーザーとしてログインする必要があります。

作成したタグに値を割り当てることもできます。値はオプションのパラメータで、タグをフィルタする際に使用できます。

手順

- 1 以下の構文を使用して要求本文を定義します。

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<tag-instance xmlns="http://www.vmware.com/vco" global="false">
  <name><tag_name></name>
  <value><tag_value></value>
</tag-instance>
```

注意 グローバル タグを作成するには、**global** 変数を **"true"** に設定します。

- 2 オブジェクトの URL で以下の POST 要求を作成します。

```
POST http://{<orchestrator_host>}:
      {<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/{<objectId>}/tags
```

POST 要求が正常に処理されると、ステータス コード 200 を受信します。

オブジェクトのタグの解除

Orchestrator REST API を使用して、オブジェクトに割り当てられているタグを削除することができます。

プライベート タグとグローバル タグはどちらも削除することができます。

注意 グローバル タグを削除するには、管理者権限を持つユーザーとしてログインする必要があります。

手順

- ◆ DELETE 要求を作成してプライベート タグまたはグローバル タグを削除します。

- プライベート タグを削除するには、オブジェクトの URL で以下の構文を使用して DELETE 要求を作成します。

```
DELETE http://{<orchestrator_host>}:
{<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/{<objectId>}/tag/{<tag_name>}
```

- グローバル タグを削除するには、オブジェクトの URL で以下の構文を使用して DELETE 要求を作成します。

```
DELETE http://{<orchestrator_host>}:
{<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/{<objectId>}/tag/{<tag_name>}
```

DELETE 要求が正常に処理されると、ステータス コード 200 を受信します。

オブジェクト タグの一覧表示

Orchestrator REST API を使用して、オブジェクトに割り当てられているタグのリストを取得することができます。

手順

- ◆ オブジェクトの URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:
{<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/{<objectId>}/tags
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。

タイプを基準にしたタグ付きオブジェクトの一覧表示

Orchestrator REST API を使用して、特定のタグが付いたオブジェクトのリストを取得し、オブジェクト タイプを基準にオブジェクトをフィルタリングできます。

手順

- ◆ オブジェクト タイプの URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/catalog/{<namespace>}/{<objectType>}/?
tags=<tag1>&tags=:<tag2>=value
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。

タグ所有者の一覧表示

Orchestrator REST API を使用して、タグの所有者を一覧表示できます。タグの所有者は、1 つ以上のタグを作成したユーザーです。

手順

- ◆ 次の URL で以下の GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/tags
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。取得した一覧には、1 つ以上のタグを作成したユーザーが表示されます。グローバル タグはシステム ユーザー名 **__GLOBAL__** の下に一覧表示されます。

ユーザーを基準にしたタグの一覧表示

Orchestrator REST API を使用して、特定のユーザーが作成したすべてのタグを取得することができます。

グローバル タグを取得することもできます。グローバル タグはシステム ユーザー名 **__GLOBAL__** の下に一覧表示されます。

手順

- ◆ ユーザーの URL で以下の GET 要求を作成します。
 - 特定のユーザーが作成したタグを一覧表示するには、次の構文に従って GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/tags/{<user_name>}
```

- グローバル タグを一覧表示するには、次の構文に従って GET 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/tags/__GLOBAL__
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。

タグ名でフィルタリングされたユーザーを基準にしたタグの一覧表示

Orchestrator REST API を使用して、特定のユーザーが作成したすべてのタグ インスタンスを取得し、タグ名を基準にオブジェクトをフィルタリングできます。

グローバル タグ インスタンスを取得することもできます。グローバル タグはシステム ユーザー名 **__GLOBAL__** の下に一覧表示されます。

手順

- ◆ ユーザーの URL で以下の **GET** 要求を作成します。
 - 特定のユーザーが作成したタグ インスタンスをフィルタリングして一覧表示するには、次の構文に従って **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/tags/{<user_name>}/{<tag_name>}
```

- グローバル タグ インスタンスをフィルタリングして一覧表示するには、次の構文に従って **GET** 要求を作成します。

```
GET http://{<orchestrator_host>}:{<port>}/vco/api/tags/__GLOBAL__/{<tag_name>}
```

GET 要求が正常に処理されると、ステータス コード 200 を受信します。取得する情報には、タグ付きオブジェクトの参照、タグ名、タグ値、およびタグ インスタンスがグローバルであるかプライベートであるかを示す情報が含まれます。

ユーザーを基準にしたタグの削除

Orchestrator REST API を使用して、特定のユーザーが作成したすべてのタグを削除できます。

グローバル タグも削除することもできます。グローバル タグはシステム ユーザー名 **__GLOBAL__** の下に一覧表示されます。

注意 グローバル タグを削除するには、管理者権限を持つユーザーとしてログインする必要があります。

手順

- ◆ ユーザーの URL で以下の **DELETE** 要求を作成します。
 - 特定のユーザーが作成したタグを削除するには、次の構文に従って **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/tags/{<user_name>}
```

- グローバル タグを削除するには、次の構文に従って **DELETE** 要求を作成します。

```
DELETE http://{<orchestrator_host>}:{<port>}/vco/api/tags/__GLOBAL__
```

DELETE 要求が正常に処理されると、ステータス コード 204 を受信します。