

# Developing a Web Services Client for VMware vRealize Orchestrator

vRealize Orchestrator 7.5



vmware®

You can find the most up-to-date technical documentation on the VMware website at:

<https://docs.vmware.com/>

If you have comments about this documentation, submit your feedback to

[docfeedback@vmware.com](mailto:docfeedback@vmware.com)

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

Copyright © 2008-2018 VMware, Inc. All rights reserved. [Copyright and trademark information.](#)

# Contents

Developing a Web Services Client for VMware vRealize Orchestrator 5

**1** Developing a Web Services Client 6

**2** Using the vRealize Orchestrator REST API 7

    Authenticating Against Orchestrator and Third-Party Systems 8

        Using vCenter Single Sign-On Authentication with the Orchestrator REST API 8

    Accessing the Reference Documentation for the Orchestrator REST API 13

    Using the Java REST SDK 13

    Operations with Workflows 14

        Find a Workflow and Retrieve Its Definition 14

        Run a Workflow 17

        Run a Workflow After Validating Its Input Parameters Against the Workflow Presentation 19

        Interacting with a Workflow While It Runs 23

        Retrieve a Workflow's Interactions 29

        Access a Workflow's Schema 30

    Working with Tasks 30

        Create a Task 30

        Modify a Task 31

        Check the State of a Task 32

    Finding Objects in the Orchestrator Inventory 33

        Find Objects by Type and ID 33

        Find Objects by Relations 34

        Apply Filters 35

    Importing and Exporting Orchestrator Objects 36

        Import a Workflow 36

        Export a Workflow 36

        Import an Action 37

        Export an Action 37

        Import a Package 38

        Export a Package 39

        Import a Resource 39

        Export a Resource 40

        Import a Configuration Element 40

        Export a Configuration Element 41

    Deleting Orchestrator Objects 41

        Delete a Workflow 41

        Delete an Action 41

Delete a Package	42
Delete a Resource	42
Delete a Configuration Element	43
Setting Permissions on Orchestrator Objects	43
REST API Permissions	43
Retrieve the Permissions of a Workflow	44
Delete the Permissions of a Workflow	44
Set the Permissions for a Workflow	45
Retrieve the Permissions of an Action	45
Delete the Permissions of an Action	46
Set the Permissions for an Action	46
Retrieve the Permissions of a Package	46
Delete the Permissions of a Package	47
Set the Permissions for a Package	47
Retrieve the Permissions of a Resource	48
Delete the Permissions of a Resource	48
Set the Permissions for a Resource	48
Retrieve the Permissions of a Configuration Element	49
Delete the Permissions of a Configuration Element	49
Set the Permissions for a Configuration Element	49
Performing Operations with Plug-Ins	50
Retrieve Information About Plug-Ins	50
Import a Plug-In	50
Export a Plug-In	51
Enable or Disable a Plug-In	51
Performing Server Configuration Operations	52
Retrieve Information About the Orchestrator Server Configuration	52
Import Orchestrator Server Configuration	52
Export Orchestrator Server Configuration	52
Performing Tagging Operations	53
Tag an Object	53
Untag an Object	54
List Object Tags	54
List Tagged Objects by Type	55
List Tag Owners	55
List Tags by Users	55
List Tags by Users Filtered by Tag Name	56
Remove Tags by Users	56

# Developing a Web Services Client for VMware vRealize Orchestrator

*Developing a Web Services Client for VMware vRealize Orchestrator* provides information about developing a Web services client for VMware® vRealize Orchestrator.

Orchestrator provides a Web services API so that you can develop applications to access and use workflows through the Web. Orchestrator provides a representational state transfer (REST) API that you can use to perform various operations over workflows.

## Intended Audience

This information is intended for Web application developers who want to access the Orchestrator processes across a network, through the RESTful Web service.

# Developing a Web Services Client



VMware vRealize Orchestrator provides Web services APIs so that you can develop applications to access workflows through the Web. The main purpose of the Orchestrator Web services APIs is to allow you to integrate Orchestrator workflows in custom Web-based applications.

Orchestrator provides a Web services API that is based on a representational state transfer (REST) API. The Orchestrator REST API exposes the objects in the Orchestrator inventory and the inventories of the installed plug-ins as resources that you can access at predefined URLs. HTTP requests at these URLs result in triggering operations over workflows. The Orchestrator REST API exposes inventory objects as resources through a set of RESTful Web services that you can use to retrieve the definitions of workflows, run workflows, check the status of the running workflows, cancel workflow runs, process waiting user interactions, retrieve the presentation of workflows, and so on.

# Using the vRealize Orchestrator REST API

# 2

The Orchestrator REST API provides functionality that allows you to communicate with the Orchestrator server directly through HTTP and perform various workflow-related operations over workflows.

The Orchestrator REST API exposes the objects from the inventories of the Orchestrator server and the installed plug-ins as resources at predefined URLs. You make HTTP calls at these URLs to trigger operations in Orchestrator. In this way, you can perform various tasks over workflows:

- Run a workflow, schedule a workflow, retrieve the runs of a workflow, answer to a user interaction, and cancel a workflow run.
- Retrieve details about a workflow such as its input and output parameters and its presentation.
- Retrieve details about a workflow run, such as its state, generated logs, start date, and end date.
- Browse the inventories of Orchestrator and the installed plug-ins.
- Import and export workflows, actions, and packages.

---

**Note** Do not use the vRealize Orchestrator REST API for installing plug-ins.

---

By using the Orchestrator REST API you can easily integrate Orchestrator workflows in custom applications that you can build in any programming language.

The Orchestrator REST API also provides eTag support as well as a mechanism for caching of response data.

This chapter includes the following topics:

- [Authenticating Against Orchestrator and Third-Party Systems](#)
- [Accessing the Reference Documentation for the Orchestrator REST API](#)
- [Using the Java REST SDK](#)
- [Operations with Workflows](#)
- [Working with Tasks](#)
- [Finding Objects in the Orchestrator Inventory](#)
- [Importing and Exporting Orchestrator Objects](#)
- [Deleting Orchestrator Objects](#)

- [Setting Permissions on Orchestrator Objects](#)
- [Performing Operations with Plug-Ins](#)
- [Performing Server Configuration Operations](#)
- [Performing Tagging Operations](#)

## Authenticating Against Orchestrator and Third-Party Systems

You must authenticate against Orchestrator in the HTTP requests that you make through the Orchestrator REST API. If you use the Orchestrator REST API to access resources on a third-party system, such as vCenter Server or vRealize Automation, you must authenticate against that system as well.

For example, to access all workflows in the Orchestrator inventory, you must authenticate against Orchestrator. However, to run a workflow against vCenter Server, you must authenticate against Orchestrator and vCenter Server.

Depending on whether you configure Orchestrator with vRealize Automation or vSphere as an authentication provider, the authentication scheme for the Orchestrator REST API is different. If Orchestrator uses vCenter Single Sign-On, depending on your configuration, you can authenticate by using a holder-of-key token issued by the vCenter Single Sign-On server. If Orchestrator is configured with vRealize Automation, you can authenticate through an OAuth bearer access token.

If you make HTTP requests at the top-level URL of the Orchestrator REST API, you do not need to authenticate against Orchestrator. The top level URL of the Orchestrator REST API is `https://orchestrator_host:port/vco/api/`.

---

**Note** The default port number for the external Orchestrator is 8281. The default port number for the Orchestrator instance that is embedded in vRealize Automation is 443.

---

A GET request at the top-level URL of the REST API returns URLs to all resources that are accessible through the API. To make HTTP requests at these URLs, you must authenticate against Orchestrator.

## Using vCenter Single Sign-On Authentication with the Orchestrator REST API

If Orchestrator is configured with the vCenter Single Sign-On Server by using the vSphere Authentication mode, you need a principal holder-of-key token to access system objects in Orchestrator through the Orchestrator REST API. To access vCenter Server or third-party systems that use the vCenter Single Sign-On Server through the Orchestrator server, you need a delegate holder-of-key token for Orchestrator and your principal token.

If Orchestrator is configured with the vCenter Single Sign-On Server, you must authenticate by using valid credentials and Orchestrator manages the holder-of-key token.

## Accessing System Objects in Orchestrator

You can access system objects in Orchestrator at the URLs of the Inventory and the Catalog services of the REST API.

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

When you access system objects in Orchestrator, you pass your principal holder-of-key token in the `Authorization` header of HTTP requests that you make to the Inventory or the Catalog service.

For example, to retrieve all system objects of type `Workflow`, you make a GET request at `https://orchestrator_host:port/vco/api/catalog/System/Workflow/`. To authenticate against Orchestrator, you need to pass your principal holder-of-key token in the `Authorization` header of the request.

## Accessing Objects in Third-Party Systems

To perform operations in third-party systems that are registered with the vCenter Single Sign-On Server through the Orchestrator REST API, you must authenticate against Orchestrator and the third-party system. You include two headers in the HTTP calls that you make through the Orchestrator REST API.

- `Authorization`. You must pass your principal holder-of-key token in this header.
- `VCOAuthorization`. You must pass a delegate holder-of-key token for Orchestrator in this header. You must acquire the delegate token for Orchestrator from the vCenter Single Sign-On Server. Orchestrator uses the delegate token to authenticate against the third-party system on your behalf.

For example, to run a workflow that uses a virtual machine through the Orchestrator REST API, you access resources both in Orchestrator and in vCenter Server. To authenticate against Orchestrator and vCenter Server, you must pass your principal holder-of-key token in the `Authorization` header of the request that you make, and the delegate token in the `VCOAuthorization` header. In this way, you authenticate against Orchestrator with your principal token and Orchestrator authenticates on your behalf against vCenter Server with the delegate token.

The vCenter Single Sign-On Server treats Orchestrator as a solution, and every solution is registered with a unique user name with the vCenter Single Sign-On Server. You request a delegate token for Orchestrator by passing the solution user name of Orchestrator and a principal holder-of-key token to the vCenter Single Sign-On Server. The token that the vCenter Single Sign-On Server issues is a delegate holder-of-key token for Orchestrator to authenticate on your behalf against third-party systems.

## Example: Obtain a Session in vCenter Single Sign-On Mode

The following example code obtains a session in vCenter Single Sign-On mode.

```
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
```

## Get the Solution User Name of Orchestrator

The vCenter Single Sign-On Server treats Orchestrator as a solution, and every solution is registered with a unique user name with the vCenter Single Sign-On Server. To be able to request a delegate holder-of-key token for Orchestrator from the vCenter Single Sign-On Server, you need the solution user name of Orchestrator.

### Prerequisites

Verify that you have a valid principal holder-of-key token that the vCenter Single Sign-On Server issued.

### Procedure

- 1 Make a GET request at the URL of the solution user name of Orchestrator:

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

- 2 Provide your principal holder-of-key token in the Authorization header of the request.

The `<user solution-user=" OrchestratorSolutionUserName"/>` element of the response contains the solution user name of Orchestrator. The following is an example of a solution user name of Orchestrator.

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

### What to do next

Use the solution user name of Orchestrator and your principal holder-of-key token to request a delegate holder-of-key token from the vCenter Single Sign-On Server.

## Using vRealize Orchestrator REST API SDK with Configured vRealize Automation Authentication

You can use REST API SDK with a configured vRA authentication in a multi-tenant or a single-tenant environment.

To obtain an authentication token (Oauth2.0) required for the code below, see the [vRO REST API authorization using Oauth2.0 Authentication \(2148518\)](#) KB article.

---

**Note** Obtain a Session in vRealize Automation Authentication Mode

---





```

"1NBQVMvdC92c3BoZXJLLmxvY2FsL2F1dGgvd2F1dGh0b2t1biIsImN0eCI6Ilt7XCJtdGRcIjpcInVybjpvYXNpczpuYW
1lczp"
+
"0YzptQU1M0jIuMDphYzpjYGFzc2VzO1Bhc3N3b3JkUHJvdGVjdGVkVHJhbnNwb3J0XCIsXCJpYXRcIjoxNTAyMjAyMTEw
LFwiwRcIjoxM"
+
"n1dIiwic2NwIjoidXNlciIsImlkciCI6IjAiLCJlbWwiOiJhZG1pbmlzdHJhdG9yQHhNLTl5LTl5LnVzIitYnUuZW
5nLnZ"
+
"td2FyZS5jb20iLCJjYWQiOiJjYXZlX2NsaS1yRlJlNEx6bURnIiwic2lkeiIjoiIiwid2lkIjoiIiwidXhwIjoxNTAyMjMw
OTEwLjYXQ0j"
+
"E1MDIyMDIxMTAsInN1YiI6IjQ1ZjQwNWUzLTNlNTgtNGJmZC1hNzMwLTQ1MjU4OWIxOGUxNyIsInByb190eXBlIjoiVVN
FUjJ9."
+ "G9gEQPtmeEH5jYab-
ILTK8NFYcwc3JZCEEjsmpUSH6oxLmZKEf-1JbsysBVH4ufqmGah3GMvmy6PUiTTamLRLfKCLwa500QCQ-qgKutZ121R"
+ "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

```

## Accessing the Reference Documentation for the Orchestrator REST API

The reference documentation for the Orchestrator REST API contains information about the RESTful Web services of the API, the data model that is applicable for the API, the response codes that are valid for the API, code examples, and so on.

The reference documentation of the Orchestrator REST API is installed together with Orchestrator. The reference documentation is available at [https://orchestrator\\_host:port/vco/api/docs/](https://orchestrator_host:port/vco/api/docs/).

The official Swagger specification is available at <https://swagger.io/specification/>.

## Using the Java REST SDK

You can use a Java SDK library to call operations on the Orchestrator REST API in Java applications and work directly with objects.

Every RESTful Web service of the Orchestrator REST SDK has a wrapping Java class with methods that correspond to the operations that can be run by using the service.

The Java REST SDK is installed together with Orchestrator. The Java REST SDK artifacts are available at the following locations.

---

**Note** You can only access the artifacts if you have deployed the Orchestrator Appliance.

---

- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client/](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-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-services/)
- [https://orchestrator\\_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/](https://orchestrator_host:port/vco-repo/com/vmware/o11n/o11n-rest-client-stubs/)

## Example: Run a Workflow and Wait for Its Completion

The following example code runs a workflow and waits for it to complete.

```
//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);
```

## Operations with Workflows

The Orchestrator REST API provides Web services that you can use to perform various operations with workflows.

### Find a Workflow and Retrieve Its Definition

To be able to perform any kind of operation with a workflow, you must find that workflow in the Orchestrator inventory and retrieve its definition. The definition lists the workflow input and output

















## Interacting with a Workflow While It Runs

The Orchestrator REST API allows you to perform various operations with a workflow during its run. You can get the status of a running workflow, answer to a waiting user interaction, and cancel a workflow run.

### Get Workflow Run Objects and Check the Workflow Status

You can get information about the runs of a workflow, such as the start and end dates, the state of the run, and the values for the input parameters. You can also get logs that are generated for a workflow run.

#### Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

#### Procedure

- 1 Retrieve the definition of the workflow whose status you want to check by making a GET request at the URL of the workflow:

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

You receive the definition of the workflow in the response body of the request. The workflow definition contains a link to the execution instances of the workflow.

- 2 Retrieve the available execution instances of the workflow by making a GET request at their URL:

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

The response body of the request lists the available execution instances of the workflow where you can view the start and end dates of every workflow run as well their status and initiator.

- 3 (Optional) To get more details about a particular run of the workflow, make a GET request at the URL of that run:

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

In the response body of the request, you receive the XML representation of the particular workflow run. You can check the values of the input parameters that are passed for this run, the user who initiated the run, the start and end dates, as well as the state of the run.

- 4 (Optional) To retrieve the logs that are generated for the workflow run, make a GET request at the URL that holds the logs:

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



## Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

## Procedure

- 1 Retrieve the list of all user interaction objects by making a GET request at the URL that holds the available user interaction objects, or by filtering only the waiting user interactions:

URL	Description
<code>https:// orchestrator_host:port/vco/api/ catalog/System/UserInteraction</code>	Holds the available user interaction objects in Orchestrator
<code>https:// orchestrator_host:port/vco/api/ catalog/System/UserInteraction? status=0</code>	Filters only the waiting user interaction objects.

You receive a list of the available user interaction objects. User interactions that are waiting have an attribute with name `state` and value `waiting`.

- 2 Make a GET request at the URL that holds the inventory item of the waiting user interaction to which you want to answer:

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

The inventory item contains a link to the user interaction instance. The user interaction instance is associated with a particular workflow run.

- 3 Make a POST request at the URL of the user interaction instance for the particular workflow execution:

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

- 4 Provide values for the input parameters of the user interaction in an `execution-context` element in the request body.

The REST API returns a 204 status when you answer to a user interaction successfully.

### Example: Answer to the User Interaction of the Interactive Hello World Workflow

You can run the Interactive Hello World sample workflow and answer to its user interaction.

- 1 Search for the waiting user interaction of the workflow by making GET request at the endpoint for the user interaction objects of the Catalog service:

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



You receive a list of the available user interaction objects. User interactions that are waiting have an attribute with name `state` and value `waiting`.

- 2 Make a GET request at the URL that holds the inventory item of the waiting user interaction that you want to answer:

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

The response body contains a link to the user interaction instance. The user interaction instance is associated with a particular workflow run.

- 3 Make a GET request at the URL of the user interaction instance:

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

In the response body, you find a down link to the presentation of the user interaction.

- 4 Make a GET request at the URL of the presentation of the user interaction:

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

You receive the definition of the user interaction presentation in the response body.

- 5 In the presentation definition, check for constraints of the values that you can pass to the input parameters.
- 6 Run the user interaction presentation by making a POST request at the URL where the instances of the presentation reside:

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

- 7 In the request body, provide values for the input parameters in an `execution-context` element.

In the response body, you receive the instance of the user interaction presentation. If the values that you passed to the input parameters are valid, you find a `valid="true"` attribute in the `execution` element. In the `output-parameters` element, you find the valid values for the input parameters that you can use to answer to the user interaction.

- 8 Answer to the user interaction by making a POST request at the URL where the user interaction instance resides:

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

- 9 In the request body, pass an `execution-context` context with the values for the input parameters.

You can use the same request body as the one for the POST request that you made at the URL for the user interaction presentation.

If the last request is successful, you receive a status code 204 and an empty response body.





- 2 Get the list of workflow interactions by making a GET request to the URL of the workflow's interactions:

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

If the GET request is successful, you receive the status code 200 and a list of all user interactions available for the workflow.

## Access a Workflow's Schema

You can access the schema image of a workflow by using the Orchestrator REST API.

### Procedure

- 1 Retrieve the definition of the workflow by making a GET request at the URL of the workflow's definition:

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

- 2 Get the workflow's schema image by making a GET request to the URL of the workflow's schema:

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

If the GET request is successful, you receive the status code 200 and the binary data of the image, representing the workflow schema. The response content type is set to a correct media type, for example Content-Type: image/png.

## Working with Tasks

Using the Task service of the Orchestrator REST API, you can perform any operation that is related to managing tasks in Orchestrator. You can create a task for scheduling a workflow, modify the properties of an already existing task, delete a task, and so on.

The maximum number of scheduled tasks supported by Orchestrator is 50.

### Create a Task

You can create a task for scheduling a workflow by using the Orchestrator REST API.

#### Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

#### Procedure

- 1 Retrieve the definition of the workflow for which you want to create a task by making a GET request at the URL of the workflow:

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



## Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

## Procedure

- 1 Make a GET request at the URL of the task that you want to modify:

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

- 2 Check the properties of the task in the response body of the request.
- 3 To modify the task, make a POST request at the URL of the task by providing the new properties of the task in a `task-data` element in the request body.

If the POST request is successful, the API reruns a status code 200 and the updated task in the response body.

## Example: Update the Send Hello Example Task

You can update the start and the end dates of a task. You can modify the example task that is introduced in [Create a Task](#). You must make a POST request at the URL of the task by providing the new start and end dates in the request body:

```
<?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>
```

## Check the State of a Task

You can check the state of the currently available tasks or check the state for all execution instances of a certain task.

## Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

## Procedure

- ◆ To check the status of all currently available tasks, make a GET request at the URL of the Task service:

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

The response body contains the definitions of the currently available tasks in Orchestrator. The state of every task is available in an attribute element, whose name is `state`. Respectively, the value for the element can be `finished`, `pending`, `running` and so on.

- ◆ To check the status of all executions of a certain task, make a GET request at the URL where the task executions reside:

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

You receive a list of the available executions for the task in the response body. The state of every execution is available in the `state` element of the task execution object.

## Finding Objects in the Orchestrator Inventory

You can find any object in the Orchestrator inventory by using the Catalog or the Inventory services. You can access only a certain subset of objects by applying filter parameters at the end of the URLs where you make HTTP requests.

You can use the Catalog service to find objects in the Orchestrator inventory that are of a certain type, or retrieve a specific object by its type and ID. For example, you can retrieve all objects that are of type `workflow` or `action`, or can retrieve a specific workflow or action.

The Inventory service allows you to browse the Orchestrator inventory by parent-child relations. Using the Inventory service, you can access objects that are available at a specific location in the Orchestrator inventory. For example, you can retrieve all workflows for Datacenter management by browsing to their location in the Orchestrator inventory, that is `Library/vCenter/Datacenter`.

Every service from the Orchestrator REST API supports filter parameters that you can add at the end of URLs when making HTTP requests. Using the filter parameters, you can narrow the results that you receive in the response body of a request at a specific URL.

## Find Objects by Type and ID

You can use the Catalog service of the REST API to find objects in Orchestrator by type and ID.

### Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

### Procedure

- 1 Make a GET request at the URL of the Catalog Service:

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



## Prerequisites

Verify that you have imported the sample workflows package in Orchestrator. The package is included in the Orchestrator sample applications ZIP file that you can download from the Orchestrator documentation page.

## Procedure

- 1 Make a GET request at the URL of the Inventory service:

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

The response body contains down links to the registered inventories of the installed plug-ins as well as to the system objects in Orchestrator under `System`.

- 2 Make a GET request at the down link of the inventory that you want to access.
- 3 Make a GET requests at the up and down links for the items in the inventory until you reach the object that you want to find.

## Example: Find the Send Hello Workflow

You can browse the Orchestrator Inventory to find the Send Hello workflow.

- 1 Make a GET request at the URL of the Inventory service:

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

- 2 Make a GET request at the URL where the system objects in Orchestrator reside:

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

- 3 Make a GET request at the URL where all workflows in Orchestrator reside:

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

- 4 Make a GET request at the URL of the Samples workflow category:

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

- 5 Use the down link for the Hello World workflow category where to locate the Send Hello workflow.

## Apply Filters

The services of the Orchestrator REST API support additional URL parameters that allow you to narrow the objects that HTTP requests to the API return.

Different query parameters are supported for every URL to a resource that you can access through the REST API. To learn which query parameters are applicable to a URL, see the *vRealize Orchestrator REST API* reference documentation.

**Procedure**

- ◆ To narrow the results from a request at a certain URL, apply filters at the end of the URL:  
*URL?filter\_1&filter\_2&filter\_3&...&filter\_N*. Every filter contains query parameters that are valid for the relevant URL. For information about the valid query parameters for every URL, see the Orchestrator REST API reference documentation.

**Example: Filter Workflows**

If you look for workflows that contain a specific word in their name, for example datastore, you can apply the following filter in a request to the Catalog Service:

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

To limit the amount of the workflows that are returned to a certain number, for example five, apply an additional filter to the request:

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

## Importing and Exporting Orchestrator Objects

The Orchestrator REST API provides Web services that you can use to import and export workflows, actions, packages, resources, and configuration elements.

### Import a Workflow

You can import a workflow by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use custom code that defines the properties of the workflow.

**Prerequisites**

The workflow binary content should be available as multi-part content. For details, see RFC 2387.

**Procedure**

- 1 In a REST client application, add request headers to define the properties of the workflow that you want to import.
- 2 Make a POST request at the URL of the workflow objects:

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

If the POST request is successful, you receive the status code 202.

### Export a Workflow

You can export a workflow by using the Orchestrator REST API and download the workflow as a file.

**Procedure**

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `accept`
  - **Value:** `application/zip`
- 2 Make a GET request at the URL of the workflow that you want to export:

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

If the GET request is successful, you receive the status code 200. The workflow binary content is available as an attachment with a default file name `workflow_name.workflow`. You can save the file with a REST client application.

**Import an Action**

You can import an action by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use custom code that defines the properties of the action.

**Prerequisites**

The action binary content should be available as multi-part content. For details, see RFC 2387.

**Procedure**

- 1 In a REST client application, add request headers to define the properties of the action that you want to import.
- 2 Make a POST request at the URL of the action objects:

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

If the POST request is successful, you receive the status code 202.

**Export an Action**

You can export an action by using the Orchestrator REST API and download the action as a file.

**Procedure**

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `accept`
  - **Value:** `application/zip`
- 2 Make a GET request at the URL of the action that you want to export:

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

If the GET request is successful, you receive the status code 200. The action binary content is available as an attachment with a default file name *action\_name.action*. You can save the file with a REST client application.

## Import a Package

You can import a package by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use custom code that defines the properties of the package.

By default, if you import an Orchestrator package with a duplicate name, the existing package is not overwritten. You can specify whether to overwrite existing packages by using a parameter in the request.

By default, Orchestrator packages are imported with the attribute values of configuration elements. You can import a package without attribute values by using a parameter in the request.

By default, tags contained in Orchestrator packages are imported, but if the same tags already exist on the Orchestrator server, the values of existing tags are preserved. You can specify whether existing tag values are preserved by using parameters in the request.

### Prerequisites

The package binary content should be available as multi-part content. For details, see RFC 2387.

### Procedure

- 1 In a REST client application, add request headers to define the properties of the package that you want to import.
- 2 Make a POST request at the URL of the package objects:

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

- 3 (Optional) To import a package and overwrite an existing package with the same name, use the `overwrite` parameter in the POST request:

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

- 4 (Optional) To import a package without the attribute values of the configuration elements from the package, use the `importConfigurationAttributeValues` parameter in the POST request:

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

- 5 (Optional) To import a package without the tags that it contains, use the `tagImportMode` parameter in the POST request:

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

- 6 (Optional) To import a package with the tags that it contains and overwrite existing tag values, use the `tagImportMode` parameter in the POST request:

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

If the POST request is successful, you receive the status code 202.

## Export a Package

You can export a package by using the Orchestrator REST API and download the package as a file.

By default, Orchestrator packages are exported with attribute values of configuration elements and global tags. You can export a package without attribute values or global tags by using parameters in the request. You can also specify a custom name for the package file that you download.

### Procedure

- In a REST client application, add a request header with the following values.
  - Name:** `accept`
  - Value:** `application/zip`
- Make a GET request at the URL of the package that you want to export:

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

- (Optional) To set a custom name for the exported package, use the `packageName` parameter in the GET request:

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

- (Optional) To export a package without the attribute values of the configuration elements from the package, use the `exportConfigurationAttributeValues` parameter in the GET request:

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

- (Optional) To export a package without global tags, use the `exportGlobalTags` parameter in the GET request:

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

If the GET request is successful, you receive the status code 200. The package binary content is available as an attachment with a default file name `package_name.package`. You can save the file with a REST client application.

## Import a Resource

You can import a resource by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use custom code that defines the properties of the resource.

### Prerequisites

The resource binary content should be available as multi-part content. For details, see RFC 2387.

### Procedure

- 1 In a REST client application, add request headers to define the properties of the resource that you want to import.
- 2 Make a POST request at the URL of the resource objects:

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

If the POST request is successful, you receive the status code 202.

## Export a Resource

You can export a resource by using the Orchestrator REST API.

### Procedure

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `accept`
  - **Value:** `application/octet-stream`
- 2 Make a GET request at the URL of the resource that you want to export:

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

If the GET request is successful, you receive the status code 200. The content of the resource is available in the response body.

## Import a Configuration Element

You can import a configuration element by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use custom code that defines the properties of the configuration element.

### Prerequisites

The configuration element binary content should be available as multi-part content. For details, see RFC 2387.

### Procedure

- 1 In a REST client application, add request headers to define the properties of the configuration element that you want to import.

- 2 Make a POST request at the URL of the configuration element objects:

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

If the POST request is successful, you receive the status code 202.

## Export a Configuration Element

You can export a configuration element by using the Orchestrator REST API.

### Procedure

- 1 In a REST client application, add a request header with the following values.

- **Name:** `accept`
- **Value:** `application/vcoobject+xml`

- 2 Make a GET request at the URL of the configuration element that you want to export:

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

If the GET request is successful, you receive the status code 200. The configuration element content is available in the response body.

## Deleting Orchestrator Objects

The Orchestrator REST API provides Web services that you can use to delete workflows, actions, packages, resources, and configuration elements.

### Delete a Workflow

You can delete a workflow by using the Orchestrator REST API.

#### Procedure

- 1 Make a GET request and retrieve the ID of the workflow from the list of returned workflows:

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

- 2 Make a DELETE request at the URL of the workflow:

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

If the DELETE request is successful, you receive the status code 200, and the response body is empty.

### Delete an Action

You can delete an action by using the Orchestrator REST API.

**Procedure**

- 1 Make a GET request and retrieve the ID of the action from the list of returned actions:

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

- 2 Make a DELETE request at the URL of the action:

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

If the DELETE request is successful, you receive the status code 200, and the response body is empty.

## Delete a Package

You can delete a package by using the Orchestrator REST API.

When you delete a package, the elements from the package are not deleted. If you want to delete the content of a package, you must provide an option parameter.

**Procedure**

- 1 Make a GET request and retrieve the name of the package from the list of returned packages:

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

- 2 Make a DELETE request at the URL of the package, and if you want to delete elements from the package, provide an option parameter at the end of the request:

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

Parameter	Description
<code>deletePackage</code>	Only the package is deleted, while its content is retained.
<code>deletePackageWithContent</code>	The package and all its content is deleted. If other packages share elements with the deleted package, the shared elements are deleted from the other packages.
<code>deletePackageKeepingShared</code>	The package and the content that is not shared is deleted. Elements that are shared with other packages are not deleted.

If you do not provide an option parameter, the default `deletePackage` parameter is used.

If the DELETE request is successful, you receive the status code 200, and the response body is empty.

## Delete a Resource

You can delete a resource by using the Orchestrator REST API.

**Procedure**

- 1 Make a GET request and retrieve the ID of the resource from the list of returned resources:

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

- 2 Make a DELETE request at the URL of the resource:

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

If the DELETE request is successful, you receive the status code 200, and the response body is empty.

## Delete a Configuration Element

You can delete a configuration element by using the Orchestrator REST API.

### Procedure

- 1 Make a GET request and retrieve the ID of the configuration element from the list of returned configuration elements:

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

- 2 Make a DELETE request at the URL of the configuration element:

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

If the DELETE request is successful, you receive the status code 200, and the response body is empty.

## Setting Permissions on Orchestrator Objects

You can set custom permissions for an Orchestrator object by using the REST API. To set the permissions, you must make a POST request at the URL of the object's permissions and define the permissions in the request body.

You can also use the Orchestrator REST API to retrieve information about an object's permissions or delete the existing permissions.

### REST API Permissions

When you set permissions by using the Orchestrator REST API, you must use a set of characters to define the permissions.

You can define the permissions for an element by including a sequence of characters in the <rights> tag of the request body of a POST request .

The characters that you can use to set permissions through the Orchestrator REST API have specific meanings.

**Table 2-1. Orchestrator REST API Permissions Character Set**

Character	Description
r	Gives view permissions.
x	Gives run permissions.
i	Gives inspect permissions.

**Table 2-1. Orchestrator REST API Permissions Character Set (continued)**

Character	Description
c	Gives edit permissions.
a	Gives administrative permissions.

### Example: Syntax for Setting Permissions

You can use the following example syntax in the request body of a POST request at the URL of an Orchestrator element's permissions.

```
<permissions xmlns="http://www.vmware.com/vco">
  <permission>
    <principal>domain.name\Group Name</principal>
    <rights>ric</rights>
  </permission>
</permissions>
```

By setting ric permissions in the <rights> tag of the request body, you allow members of your user group to view, inspect, and edit the Orchestrator element.

### Retrieve the Permissions of a Workflow

You can retrieve information about the permissions of a workflow by using the Orchestrator REST API.

#### Procedure

- 1 Make a GET request and retrieve the ID of the workflow from the list of returned workflows:

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

- 2 Make a GET request at the URL of the workflow's permissions:

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

If the GET request is successful, you receive the status code 200. Information about the workflow's permissions is available in the response body.

### Delete the Permissions of a Workflow

You can delete the permissions of a workflow by using the Orchestrator REST API. You can delete the existing permissions of a workflow before you set new permissions.

#### Procedure

- 1 Make a GET request and retrieve the ID of the workflow from the list of returned workflows:

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

- 2 Make a DELETE request at the URL of the workflow's permissions:

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

If the DELETE request is successful, you receive the status code 204, and the response body is empty.

## Set the Permissions for a Workflow

You can set the permissions for a workflow by using the Orchestrator REST API.

### Prerequisites

Review the types of permissions that you can set and the syntax that you can use in the request body. See [REST API Permissions](#).

### Procedure

- 1 Make a GET request and retrieve the ID of the workflow from the list of returned workflows:

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

- 2 In a REST client application, add request headers to define the properties of the workflow for which you want to set permissions.
- 3 In the request body, specify the permissions that you want to set.
- 4 Make a POST request at the URL of the workflow's permissions:

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

If the POST request is successful, you receive the status code 201. Information about the workflow's permissions is available in the response body.

## Retrieve the Permissions of an Action

You can retrieve information about the permissions of an action by using the Orchestrator REST API.

### Procedure

- 1 Make a GET request and retrieve the ID of the action from the list of returned actions:

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

- 2 Make a GET request at the URL of the action's permissions:

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

If the GET request is successful, you receive the status code 200. Information about the action's permissions is available in the response body.

## Delete the Permissions of an Action

You can delete the permissions of an action by using the Orchestrator REST API. You can delete the existing permissions of an action before you set new permissions.

### Procedure

- 1 Make a GET request and retrieve the ID of the action from the list of returned actions:

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

- 2 Make a DELETE request at the URL of the action's permissions:

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

If the DELETE request is successful, you receive the status code 204, and the response body is empty.

## Set the Permissions for an Action

You can set the permissions for an action by using the Orchestrator REST API.

### Prerequisites

Review the types of permissions that you can set and the syntax that you can use in the request body. See [REST API Permissions](#).

### Procedure

- 1 Make a GET request and retrieve the ID of the action from the list of returned actions:

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

- 2 In a REST client application, add request headers to define the properties of the action for which you want to set permissions.
- 3 In the request body, specify the permissions that you want to set.
- 4 Make a POST request at the URL of the action's permissions:

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

If the POST request is successful, you receive the status code 201. Information about the action's permissions is available in the response body.

## Retrieve the Permissions of a Package

You can retrieve information about the permissions of a package by using the Orchestrator REST API.

### Procedure

- 1 Make a GET request and retrieve the name of the package from the list of returned packages:

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

- 2 Make a GET request at the URL of the package's permissions:

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

If the GET request is successful, you receive the status code 200. Information about the package's permissions is available in the response body.

## Delete the Permissions of a Package

You can delete the permissions of a package by using the Orchestrator REST API. You can delete the existing permissions of a package before you set new permissions.

### Procedure

- 1 Make a GET request and retrieve the name of the package from the list of returned packages:

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

- 2 Make a DELETE request at the URL of the package's permissions:

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

If the DELETE request is successful, you receive the status code 204, and the response body is empty.

## Set the Permissions for a Package

You can set the permissions for a package by using the Orchestrator REST API.

### Prerequisites

Review the types of permissions that you can set and the syntax that you can use in the request body. See [REST API Permissions](#).

### Procedure

- 1 Make a GET request and retrieve the name of the package from the list of returned packages:

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

- 2 In a REST client application, add request headers to define the properties of the package for which you want to set permissions.
- 3 In the request body, specify the permissions that you want to set.
- 4 Make a POST request at the URL of the package's permissions:

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

If the POST request is successful, you receive the status code 201. Information about the package's permissions is available in the response body.

## Retrieve the Permissions of a Resource

You can retrieve information about the permissions of a resource by using the Orchestrator REST API.

### Procedure

- 1 Make a GET request and retrieve the ID of the resource from the list of returned resources:

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

- 2 Make a GET request at the URL of the resource's permissions:

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

If the GET request is successful, you receive the status code 200. Information about the resource's permissions is available in the response body.

## Delete the Permissions of a Resource

You can delete the permissions of a resource by using the Orchestrator REST API. You can delete the existing permissions of a resource before you set new permissions.

### Procedure

- 1 Make a GET request and retrieve the ID of the resource from the list of returned resources:

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

- 2 Make a DELETE request at the URL of the resource's permissions:

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

If the DELETE request is successful, you receive the status code 204, and the response body is empty.

## Set the Permissions for a Resource

You can set the permissions for a resource by using the Orchestrator REST API.

### Prerequisites

Review the types of permissions that you can set and the syntax that you can use in the request body. See [REST API Permissions](#).

### Procedure

- 1 Make a GET request and retrieve the ID of the resource from the list of returned resources:

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

- 2 In a REST client application, add request headers to define the properties of the resource for which you want to set permissions.
- 3 In the request body, specify the permissions that you want to set.

- 4 Make a POST request at the URL of the resource's permissions:

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

If the POST request is successful, you receive the status code 201. Information about the resource's permissions is available in the response body.

## Retrieve the Permissions of a Configuration Element

You can retrieve information about the permissions of a configuration element by using the Orchestrator REST API.

### Procedure

- 1 Make a GET request and retrieve the ID of the configuration element from the list of returned configuration elements:

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

- 2 Make a GET request at the URL of the configuration element's permissions:

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

If the GET request is successful, you receive the status code 200. Information about the configuration element's permissions is available in the response body.

## Delete the Permissions of a Configuration Element

You can delete the permissions of a configuration element by using the Orchestrator REST API. You can delete the existing permissions of a configuration element before you set new permissions.

### Procedure

- 1 Make a GET request and retrieve the ID of the configuration element from the list of returned configuration elements:

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

- 2 Make a DELETE request at the URL of the configuration element's permissions:

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

If the DELETE request is successful, you receive the status code 204, and the response body is empty.

## Set the Permissions for a Configuration Element

You can set the permissions for a configuration element by using the Orchestrator REST API.

## Prerequisites

Review the types of permissions that you can set and the syntax that you can use in the request body. See [REST API Permissions](#).

## Procedure

- 1 Make a GET request and retrieve the ID of the configuration element from the list of returned configuration elements:

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

- 2 In a REST client application, add request headers to define the properties of the configuration element for which you want to set permissions.
- 3 In the request body, specify the permissions that you want to set.
- 4 Make a POST request at the URL of the configuration element's permissions:

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

If the POST request is successful, you receive the status code 201. Information about the configuration element's permissions is available in the response body.

# Performing Operations with Plug-Ins

The Orchestrator REST API provides Web services that you can use to perform various operations with plug-ins.

## Retrieve Information About Plug-Ins

You can retrieve metadata information for all installed plug-ins by using the Orchestrator REST API.

### Procedure

- 1 In a REST client application, add request headers to define the properties of the plug-ins.
- 2 Make a GET request at the URL of the plug-in objects:

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

If the GET request is successful, you receive the status code 200.

## Import a Plug-In

You can import a plug-in by using the Orchestrator REST API.

Depending on the library of your REST client application, you can use a custom code that defines the properties of the plug-in.

---

**Note** You cannot import a plug-in if a plug-in with the same name is already installed.

---

## Prerequisites

The plug-in binary content should be available as multi-part content. For details, see RFC 2387.

## Procedure

- 1 In a REST client application, add request headers to define the properties of the plug-in that you want to import.
- 2 Make a POST request at the URL of the plug-in objects:

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

If the POST request is successful, you receive the status code 200.

## Export a Plug-In

You can export a plug-in by using the Orchestrator REST API.

## Procedure

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `accept`
  - **Value:** `application/dar`
- 2 Make a GET request at the URL of the plug-in that you want to export:

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

If the GET request is successful, you receive the status code 200. The plug-in content is available in the response body.

## Enable or Disable a Plug-In

You can enable or disable a plug-in by using the Orchestrator REST API.

You can change the state of a plug-in from enabled to disabled, or from disabled to enabled, by making a PUT request at the URL of the plug-in. You can check the current state of a plug-in by retrieving information about the Orchestrator plug-ins. See [Retrieve Information About Plug-Ins](#).

## Prerequisites

The plug-in binary content should be available as multi-part content. For details, see RFC 2387.

## Procedure

- 1 In a REST client application, add request headers to define the properties of the plug-in that you want to enable or disable.
- 2 Make a PUT request at the URL of the plug-in that you want to enable or disable:

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

If the PUT request is successful, you receive the status code 200.

## Performing Server Configuration Operations

The Orchestrator REST API provides Web services that you can use to perform various operations related to the Orchestrator server configuration.

### Retrieve Information About the Orchestrator Server Configuration

You can retrieve information about the Orchestrator server configuration by using the Orchestrator REST API.

#### Procedure

- 1 In a REST client application, add request headers to define the properties of the server for which you want to retrieve information.
- 2 Make a GET request at the URL of the plug-in objects:

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

If the GET request is successful, you receive the status code 200.

### Import Orchestrator Server Configuration

You can import a saved configuration by using the Orchestrator REST API.

#### Prerequisites

The configuration binary content should be available as multi-part content. For details, see RFC 2387.

#### Procedure

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `content-type`
  - **Value:** `multipart/form-data`
- 2 Make a POST request at the URL of the server configuration:

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

If the POST request is successful, you receive the status code 200.

### Export Orchestrator Server Configuration

You can export the server configuration by using the Orchestrator REST API.

#### Prerequisites

The configuration binary content should be available as multi-part content. For details, see RFC 2387.

## Procedure

- 1 In a REST client application, add a request header with the following values.
  - **Name:** `content-type`
  - **Value:** `multipart/form-data`
- 2 Add another request header with the following values.
  - **Name:** `accept`
  - **Value:** `*/*`
- 3 Make a POST request at the URL of the server configuration:

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

If the POST request is successful, you receive the status code 200.

## Performing Tagging Operations

The Orchestrator REST API provides Web services that you can use to perform various operations to make objects more searchable by using tags in Orchestrator.

You can make objects more searchable by attaching tags to them. Tags are strings with length between 3 and 64 characters and must contain no whitespace characters.

You can add global and private tags. Global tags are visible to all Orchestrator users and private tags are visible only to the user who created them. Global tags can be created and removed only by users with administrative privileges.

### Tag an Object

You can assign tags to an object by using the Orchestrator REST API.

You can create both private and global tags. You specify whether the tag is private or global in the body of the request.

---

**Note** To create global tags, you must be logged in as a user with administrative privileges.

---

You can also assign a value to the tag that you create. The value is an optional parameter that you can use to filter tags.

## Procedure

- 1 Define the request body by using the following syntax.

```
<?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>
```

---

**Note** You can create a global tag by setting the `global` variable to `"true"`.

---

- 2 Make a POST request at the URL of the object:

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

If the POST request is successful, you receive the status code 200.

## Untag an Object

You can remove tags assigned to an object by using the Orchestrator REST API.

You can remove both private and global tags.

---

**Note** To remove global tags, you must be logged in as a user with administrative privileges.

---

## Procedure

- ◆ Make a DELETE request to remove private or global tags.
  - To remove a private tag, make a DELETE request at the URL of the object by using the following syntax:

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

- To remove a global tag, make a DELETE request at the URL of the object by using the following syntax:

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

If the DELETE request is successful, you receive the status code 200.

## List Object Tags

You can retrieve a list of tags assigned to an object by using the Orchestrator REST API.

**Procedure**

- ◆ Make a GET request at the URL of the object:

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

If the GET request is successful, you receive the status code 200.

**List Tagged Objects by Type**

You can use the Orchestrator REST API to retrieve a list of objects tagged with a specific tag and filter them by object type.

**Procedure**

- ◆ Make a GET request at the URL of the object type:

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

If the GET request is successful, you receive the status code 200.

**List Tag Owners**

You can retrieve a list of tag owners by using the Orchestrator REST API. Tag owners are users who have created at least one tag.

**Procedure**

- ◆ Make a GET request at the following URL:

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

If the GET request is successful, you receive the status code 200. The list that you retrieve contains users who have created at least one tag. Global tags are listed under the system user name `__GLOBAL__`.

**List Tags by Users**

You can use the Orchestrator REST API to retrieve a list of tags created by a specific user.

You can also retrieve global tags. Global tags are listed under the system user name `__GLOBAL__`.

**Procedure**

- ◆ Make a GET request at the URL of the user.
  - To retrieve a list of the tags created by a specific user, make a GET request by using the following syntax:

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

- To retrieve a list of global tags, make a GET request by using the following syntax:

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

If the GET request is successful, you receive the status code 200.

## List Tags by Users Filtered by Tag Name

You can use the Orchestrator REST API to retrieve a list of tag instances created by a specific user and filter the tags by tag name.

You can also retrieve global tag instances. Global tags are listed under the system user name \_\_GLOBAL\_\_.

### Procedure

- ◆ Make a GET request at the URL of the user.
  - To retrieve a filtered list of the tag instances created by a specific user, make a GET request by using the following syntax:

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

- To retrieve a filtered list of global tag instances, make a GET request by using the following syntax:

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

If the GET request is successful, you receive the status code 200. The information that you retrieve contains a reference to the tagged object, tag name, tag value, and an indication whether the tag instance is global or private.

## Remove Tags by Users

You can use the Orchestrator REST API to remove all tags created by a specific user.

You can also remove global tags. Global tags are listed under the system user name \_\_GLOBAL\_\_.

---

**Note** To remove global tags, you must be logged in as a user with administrative privileges.

---

### Procedure

- ◆ Make a DELETE request at the URL of the user.
  - To remove the tags created by a specific user, make a DELETE request by using the following syntax:

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

- To remove the global tags, make a DELETE request by using the following syntax:

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

If the DELETE request is successful, you receive the status code 204.