VMware vCloud Suite SDK for Java Programming Guide

Update 1
vCloud Suite SDK for Java 6.0
vCenter Server 6.0
VMware ESXi 6.0

This document supports the version of each product listed and supports all subsequent versions until the document is replaced by a new edition. To check for more recent editions of this document, see http://www.vmware.com/support/pubs.

EN-001848-02

vmware®

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

http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

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

docfeedback@vmware.com

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

Contents

About VMware vCloud Suite SDK for Java Programming Guide

VMware vCloud Suite SDK for Java Programming Guide describes how to use the VMware vCloud Suite SDK for Java. VMware provides different APIs and SDKs for different applications and goals. The vCloud Suite SDK for Java supports the development of clients that use the vCloud Suite SDK for infrastructure support tasks .

Intended Audience

This information is intended for anyone who will develop applications by using the vCloud Suite SDK for Java. Some programming background in Java is required

Updated Information

This *VMware vCloud Suite SDK for Java Programming Guide* is updated with each release of the product or when necessary.

This table provides the update history of the *VMware vCloud Suite SDK for Java Programming Guide*.

Revision	Description
EN-001848-02	Removed incorrect references to authenticating individual operations with a token. See, for example, Chapter 4, "Authentication Mechanisms," on page 25
EN-001848-01	Restructured initial chapters for clarity.
EN-001848-00	Initial release.

Introduction to the vCloud Suite SDKs

1

The vCloud Suite SDKs bundle client libraries for accessing new vCloud Suite features like Content Library and existing features like Tagging. The vCloud Suite SDKs contain sample applications and API reference documentation for the Content Library and Tagging services. The vCloud Suite SDKs also provide sample code that retrieves the endpoints of vCloud Suite and vSphere services and establishes a secure connection with the vCloud Suite endpoint.

vCloud Suite supports six languages for accessing the vCloud Suite API services and provides six SDKs for developing client applications for managing components in your virtual environment.

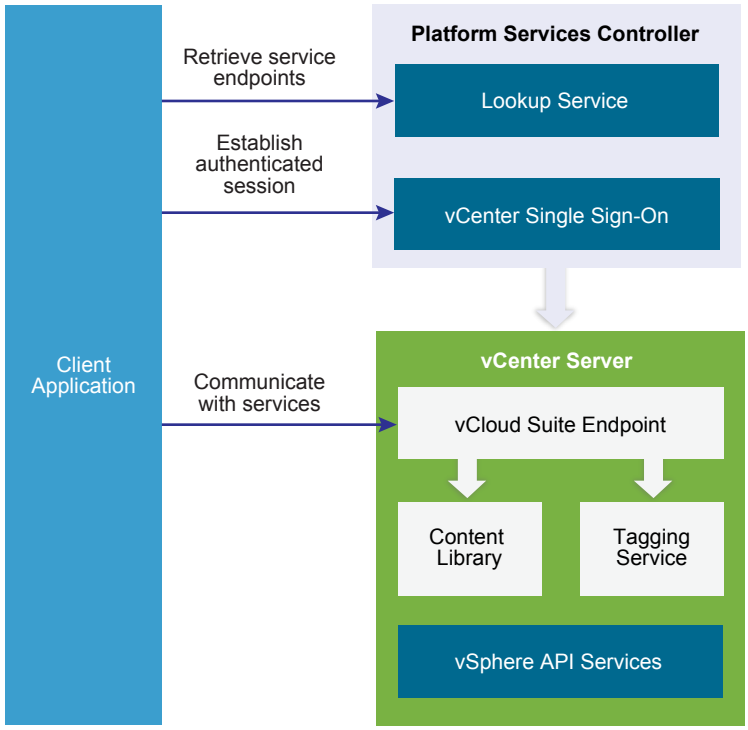
This chapter includes the following topics:

vCloud Suite SDKs Overview

The vCloud Suite API provides a unified programming interface to vCloud Suite services that you can use through SDKs provided in six programming languages. The vCloud Suite API provides a service-oriented architecture for accessing resources in the virtual environment by issuing requests to the vCloud Suite Endpoint.

vCloud Suite API client applications communicate with services on the Platform Services Controller and vCenter Server.

Figure 1-1. Communication Model of vCloud Suite API Client Applications



vCloud Suite API client applications use the Lookup Service to retrieve the vCenter Single Sign-On endpoint, the vCloud Suite Endpoint, and the endpoints of services that are exposed through the vSphere API. To access vCloud Suite services such as Content Library and Tagging, client applications issue requests to the vCloud Suite Endpoint. By using the vCenter Single Sign-On service, client applications can either establish an authenticated vCloud Suite session, or authenticate individual requests to the vCloud Suite Endpoint.

Client applications can access services that are exposed through the vSphere API by using the vSphere Management SDK.

Depending on the vSphere deployment model, client applications can communicate with vCloud Suite services on a single vCenter Server instance or multiple vCenter Server instances. For more information about the vSphere deployment models, see Chapter 2, "Components of the vCloud Suite Virtualization Layer," on page 13

SDK Developer Setup

To start developing a vCloud Suite API client application, you must download the software and set up a development environment. You can find instructions for setting up a development environment in the README for each vCloud Suite SDK.

SDK Samples

The vCloud Suite SDKs provide sample applications that you can extend to implement client applications that serve your needs. The code examples in the vCloud Suite SDKs documentation are based on these sample applications.

Supported Programming Languages

The vCloud Suite SDKs are packed in six different programming languages that let you build client applications on your preferred programming language.

- vCloud Suite SDK for Java
- vCloud Suite SDK for Python
- vCloud Suite SDK for .NET
- vCloud Suite SDK for Perl
- vCloud Suite SDK for Ruby
- vCloud Suite SDK for REST

Components of the vCloud Suite Virtualization Layer 2

At the core of vCloud Suite is vSphere, which provides the virtualization layer of the software-defined data center. You can use vSphere deployment options for vCenter Server and ESXi hosts to build virtual environments of different scales.

This chapter includes the following topics:

Components and Services of a Virtual Environment

Starting with vSphere 6.0, the deployment of the virtual environment consists of two major components that provide different sets of services, the VMware Platform Services Controller and vCenter Server. You can deploy vCenter Server with an embedded or external Platform Services Controller.

Services Installed with Platform Services Controller

The Platform Services Controller group of infrastructure services contains vCenter Single Sign-On, License Service, Lookup Service, and VMware Certificate Authority. The services installed with the Platform Services Controller are common to the entire virtual environment. A Platform Services Controller can be connected to one or more vCenter Server instances. In a deployment that consists of more than one Platform Services Controller, the data of each service is replicated across all Platform Services Controller instances.

In vCloud Suite API client applications, you use the vCenter Single Sign-On and the Lookup Service on the Platform Services Controller to provide a range of functionality.

- Authentication and session management. You use the vCenter Single Sign-On service to establish an authenticated session with the vCloud Suite Endpoint. You send credentials to the vCenter Single Sign-On service and receive a SAML token that you use to obtain a session ID from the vCloud Suite Endpoint. Alternatively, you can access the vCloud Suite APIs in a sessionless manner by including the SAML token in every request that you issue to the vCloud Suite Endpoint.
- Service discovery. You use the Lookup Service to discover the endpoint URL for the vCenter Single Sign-On service on the Platform Services Controller, the location of the vCenter Server instances, and the vCloud Suite Endpoint.

Services Installed with vCenter Server

vCenter Server is a central administration point for ESXi hosts. The vCenter Server group of services contains vCenter Server, vSphere Web Client, Inventory Service, vSphere® Auto Deploy™, vSphere® ESXi™ Dump Collector, VMware vSphere® Syslog Collector on Windows and VMware vSphere Syslog Service for the VMware vCenter Server™ Appliance™.

vCenter Server also provides services that you can access through the vCloud Suite Endpoint.

Content Library Service You can use the Content Library Service to share VM templates, vApp templates, and other files across the software-defined data center. You can create, share, and subscribe to content libraries on the same vCenter Server instance or on a remote instance. This promotes consistency, compliance, efficiency, and automation in deploying workloads at scale. By using content libraries, you can also create OVF packages from virtual machines and virtual appliances in hosts, resource pools, and clusters. You can then use the OVF packages to provision new virtual machines in hosts, resource pools, and clusters.

Tagging Service This service supports the definition of tags that you can associate with vSphere objects or vCloud Suite resources. The vCloud Suite SDKs provide the capability to manage tags programmatically.

vSphere Deployment Configurations

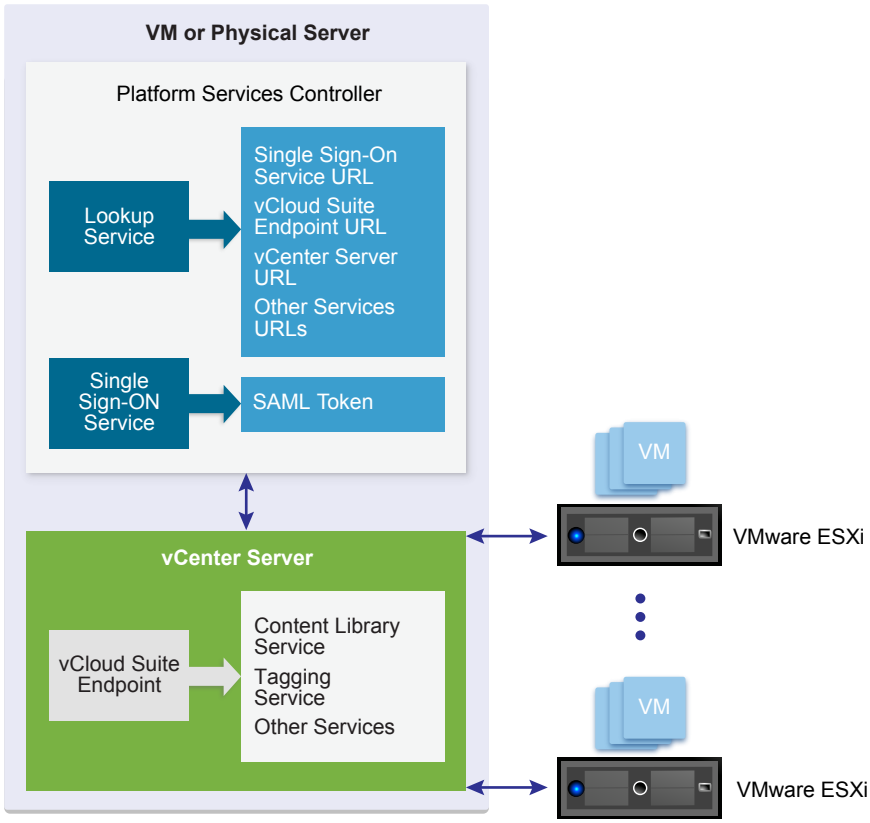
vCloud Suite client applications communicate with services on the Platform Services Controller and vCenter Server components of the virtual environment. vCenter Server can be deployed with an embedded or external Platform Services Controller.

vCenter Server with an Embedded Platform Services Controller

vCenter Server and Platform Services Controller reside on the same virtual machine or physical server. This deployment is most suitable for small environments such as development or test beds.

A client application must first connect to the Lookup Service on the Platform Services Controller and retrieve the vCenter Single Sign-On Service endpoint and the vCloud Suite Endpoint. The application can access services on the vCenter Server instance through the vCloud Suite Endpoint in an authenticated session or by providing as SAML token in each request.

Figure 2-1. vCenter Server with Embedded Platform Services Controller

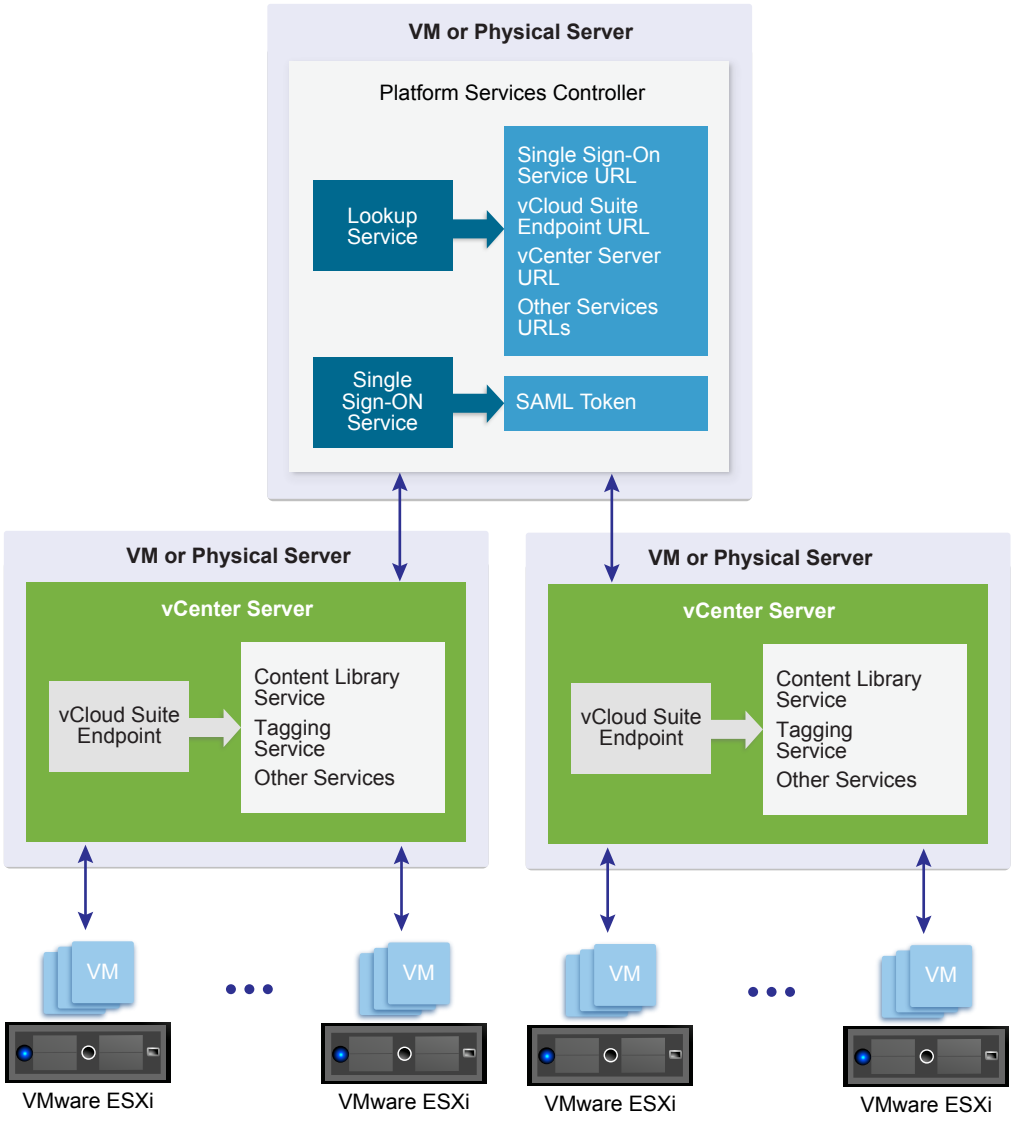


vCenter Server with an External Platform Services Controller

vCenter Server and Platform Services Controller are deployed on separate virtual machines or physical servers. The Platform Services Controller can be shared across several vCenter Server instances. For larger deployments, more than one Platform Services Controller can exist as infrastructure data is replicated across all Platform Services Controller instances.

A client application functions in a similar way as in a Platform Services Controller with embedded vCenter Server deployment. The only difference is that the client application can access services on multiple vCenter Server instances, or services only on a particular vCenter Server instance.

Figure 2-2. vCenter Server with External Platform Services Controller



Retrieving Service Endpoints 3

To access services and resources in the virtual environment, vCloud Suite API client applications must retrieve endpoints of vCloud Suite and vSphere services. Client applications retrieve service endpoints from the Lookup Service that runs on the Platform Services Controller.

The Lookup Service provides service registration and discovery by using the vSphere Web services API. By using the Lookup Service, you can retrieve endpoints of services on the Platform Services Controller and vCenter Server.

- The vCenter Single Sign-On endpoint on the Platform Services Controller. You use the vCenter Single Sign-On service to get a SAML token and establish an authenticated session with the vCloud Suite Endpoint. You can also use the token in a sessionless manner by including it in every request that you issue to the vCloud Suite Endpoint. For more information, see Chapter 4, "Authentication Mechanisms," on page 25.
- The vCloud Suite Endpoint on vCenter Server. Through the vCloud Suite Endpoint, you can make requests to vCloud Suite API services such as Content Library and Tagging.
- The vCenter Server endpoint. In an environment with external Platform Services Controller instances, you can use the vCenter Server endpoint to get the node ID of a particular vCenter Server instance. By using the node ID , you can retrieve service endpoints on that vCenter Server instance.
- The vSphere API endpoint and endpoints of other vSphere services that run on vCenter Server.

Workflow for Retrieving Service Endpoints

The workflow that you use to retrieve service endpoints from the Lookup Service might vary depending on the endpoints that you need and their number. Follow a general workflow for retrieving service endpoints.

1. Connect to the Lookup Service on the Platform Services Controller and service registration object so that you can query for registered services.
2. Create a service registration filter for the endpoints that you want to retrieve.
3. Use the filter to retrieve registration information for services from the Lookup Service.
4. Extract one or more endpoint URLs from the array of registration information that you receive from the Lookup Service.

This chapter includes the following topics:

- "Retrieve a vCenter Server ID by Using the Lookup Service," on page 23
- "Retrieve a vCloud Suite Endpoint on a vCenter Server Instance," on page 24

Filtering for Predefined Service Endpoints

The Lookup Service maintains a registration list of all services in the virtual environment. You can use the Lookup Service to retrieve registration information for every service by setting a registration filter that you pass to the `List()` function on the Lookup Service. The functions and objects that you can use with the Lookup Service are defined in the `lookup.wsdl` file that is part of the SDK.

Lookup Service Registration Filters

You can query for service endpoints through a service registration object that you obtain from the Lookup Service. You invoke the`List()` function on the Lookup Service to list the endpoints that you need by passing `LookupServiceRegistrationFilter` . `LookupServiceRegistrationFilter` identifies the service and the endpoint type that you can retrieve.

Optionally, you can include the node ID parameter in the filter that identifies the vCenter Server instance where the endpoint is hosted. When the node ID is omitted, the `List()` function returns the endpoint URLs of the service that are hosted on all vCenter Server instances in the environment.

For example, a `LookupServiceRegistrationFilter` for queering the vCloud Suite service has these service endpoint elements.

Table 3-1. Service Registration Filter Parameters

Filter Types	Value	Description
`LookupServiceRegistrationServiceType`	product= "com.vmware.cis"	vCloud Suite namespace.
	type= "cs.vapi"	Identifies the vCloud Suite service.
`LookupServiceRegistrationEndpointType`	type= "com.vmware.vapi.endpoint"	Specifies the endpoint path for the service.
	protocol= "vapi.json.https.public"	Identifies the protocol that will be used for communication with the endpoint .

For information about the filter parameter of the available predefined service endpoints, see "Filter Parameters for Predefined Service Endpoints," on page 18.

Filter Parameters for Predefined Service Endpoints

Depending on the service endpoint that you want to retrieve, you provide different parameters to the `LookupServiceRegistrationFilter` that you pass to the `List()` function on the Lookup Service. To search for services on a particular vCenter Server instance, set the node ID parameter to the filter.

Table 3-2. Input Data for URL Retrieval for the Lookup Service Registration Filter

Service	Input Data	Value
vCenter Single Sign-On	product namespace	`com.vmware.cis`
	service type	`cs.identity`
	protocol	`wsTrust`
	endpoint type	`com.vmware.cis.cs.identity.sso`

Table 3-2. Input Data for URL Retrieval for the Lookup Service Registration Filter (Continued)

Service	Input Data	Value
vCloud Suite Endpoint	product namespace	`com.vmware.cis`
	service type	`cs.vapi`
	protocol	`vapi.json.https.public`
	endpoint type	`com.vmware.vapi.endpoint`
vCenter Server	product namespace	`com.vmware.cis`
	service type	`vcenterserver`
	protocol	`vmomi`
	endpoint type	`com.vmware.vim`
vCenter Storage Monitoring Service	product namespace	`com.vmware.vim.sms`
	service type	`sms`
	protocol	`https`
	endpoint type	`com.vmware.vim.sms`
vCenter Storage Policy-Based Management	product namespace	`com.vmware.vim.sms`
	service type	`sms`
	protocol	`https`
	endpoint type	`com.vmware.vim.pbm`
vSphere ESX Agent Manager	product namespace	`com.vmware.vim.sms`
	service type	`cs.eam`
	protocol	`vmomi`
	endpoint type	`com.vmware.cis.cs.eam.sdk`

Connect to the Lookup Service and Retrieve the Service Registration Object

You must connect to the Lookup Service to gain access to its operations. After you connect to the Lookup Service, you must retrieve the service registration object to make registration queries.

Procedure

1 Connect to the Lookup Service.

   a Configure a connection stub for the Lookup Service endpoint, which uses SOAP bindings, by using the HTTPS protocol.

   b Create a connection object to communicate with the Lookup Service.

2 Retrieve the Service Registration Object.

   a Create a managed object reference.

   b Retrieve the `ServiceContent` managed object.

   c Retrieve the service registration object.

   With the service registration object, you can make registration queries.

Java Example of Connecting to the Lookup Service and Retrieving the Service Registration Object

The example is based on the code in the `LookupServiceHelper.java` sample file. The file is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/vmware/vcloud/suite/samples/common/`

NOTE The connection code in the example disables certificate and host name checking for the connection for simplicity. For a production deployment, supply appropriate handlers. See the SDK sample file for a more detailed example of connection code.

```
import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.HttpsURLConnection;
import javax.net.ssl.SSLSession;
import javax.xml.ws.BindingProvider;

import com.vmware.vcloud.suite.lookup.LsService;
import com.vmware.vcloud.suite.lookup.LsPortType;
import com.vmware.vcloud.suite.lookup.ManagedObjectReference;
import com.vmware.vcloud.suite.lookup.LookupServiceContent;

...

  String lookupServiceUrl;
  LsService lookupService;
  LsPortType lsPort;
  ManagedObjectReference serviceInstanceRef;
  LookupServiceContent lookupServiceContent;
  ManagedObjectReference serviceRegistration;

//1 – Configure Lookup Service stub.
  HostnameVerifier hostVerifier = new HostnameVerifier (){
    public boolean verify(String urlHostName, SSLSession session){
     return true;
    }
  };

  HttpsURLConnection.setDefaultHostnameVerifier(hostVerifier);
  SslUtil.trustAllHttpsCertificates();


//2 – Create the Lookup Service stub.
  lookupService = new LsService();
  lsPort = new LsPorType.getLsPort();

((BindingProvider)lsProvider).getRequestContext().put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY,
lookupServiceUrl);

//4 – Create a predetermined management object.
  serviceInstanceRef = new ManagedObjectReference();
  serviceInstanceRef.setType("LookupServiceInstance");
  serviceInsanceRefl.setValue("ServiceInstance");

//5 – Retrieve the ServiceContent object.
```

```
  lookupServiceContent = lsPort.retrieveServiceContent(serviceInstanceRef);

//6 - Retrieve the service registration
  serviceRegistration = lookupServiceContent.getServiceRegistration();

...
```

Retrieve Service Endpoints on vCenter Server Instances

You can create a function that obtains the endpoint URLs of a service on all vCenter Server instances in the environment. You can modify that function to obtain the endpoint URL of a service on a particular vCenter Server instance.

See "Connect to the Lookup Service and Retrieve the Service Registration Object," on page 19.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

Procedure

1 Create a filter criterion for service information.

2 Create a filter criterion for endpoint information.

3 Create the registration filter object.

Option	Description
Omit the node ID parameter	Retrieves the endpoint URLs of the service on all vCenter Server instances.
Include the node ID parameter	Retrieves the endpoint URL of the service on a particular vCenter Server instance.

4 Retrieve the specified service information by using the `List()` function.

Depending on whether you included the node ID parameter, the `List()` function returns different results.

- A list of endpoint URLs for a service that are is hosted on all vCenter Server instances in the environment.
- An endpoint URL of a service that runs on a particular vCenter Server.

What to do next

Call the function that you implemented to retrieve service endpoints. You can pass different filter parameters depending on the service endpoints that you need. For more information, see "Filter Parameters for Predefined Service Endpoints," on page 18.

To retrieve a service endpoint on a particular vCenter Server instance, you must retrieve the node ID of that instance and pass it to the function. For information about how to retrieve the ID of a vCenter Server instance, see "Retrieve a vCenter Server ID by Using the Lookup Service," on page 23.

Java Example of Retrieving a Service Endpoint on a vCenter Server Instance

This example provides a common pattern for filtering Lookup Service registration data. This example is based on the code in the `LookupServiceHelper.java` sample file. The file is located in the following vCloud Suite SDK for Java directory: `client/samples/java/vmware/vcloud/suite/samples/common/`

```
...
import com.vmware.vcloud.suite.lookup.LookupServiceRegistrationEndpointType;
import com.vmware.vcloud.suite.lookup.LookupServiceRegistrationFilter;
import com.vmware.vcloud.suite.lookup.LookupServiceRegistrationServiceType;

...

/**
 * Define filter criterion for retrieving a service endpoint.
 * Omit the nodeID parameter to retrieve the endpoints hosted
 * on all vCenter Server instances in the environment.
 */

  List<LookupServiceRegistrationInfo> lookupServiceUrls(String prod,
                                                        String svcType,
                                                        String proto,
                                                        String epType,
                                                        String nodeID){

    LookupServiceRegistrationServiceType filterServiceType =
                                                       new LookupServiceRegistrationServiceType();
    filterServiceType.setProduct(prod);
    filterServiceType.setType(svcType);

    LookupServiceRegistrationEndpointType filterEndpointType =
                                                       new LookupServiceRegistrationEndpointType();
    filterEndpointType.setProtocol(proto);
    filterEndpointType.setType(epType);

    LookupServiceRegistrationFilter filterCriteria = new LookupServiceRegistrationFilter();
    filterCriteria.setServiceType(filterServiceType);
    filterCriteria.setEndpointType(filterEndpointType);
    filterCriteria.setNode(nodeID);

    return lsPort.list(serviceRegistration, filterCriteria);
  }

...
```

Retrieve a vCenter Server ID by Using the Lookup Service

You use the node ID of a vCenter Server instance to retrieve the endpoint URL of a service on that vCenter Server instance. You specify the node ID in the service registration filter that you pass to the `List()` function on the Lookup Service.

Managed services are registered with the instance name of the vCenter Server instance where they run. The instance name maps to a unique vCenter Server ID. The instance name of a vCenter Server system is specified during installation and might be an FQDN or an IP address.

NOTE Use the URL that you retrieve from the Lookup Service to access a service. Avoid using the instance name to construct the URL of the service.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.

Procedure

1 List the vCenter Server instances.

2 Find the matching node name of the vCenter Server instance and save the ID.

Use the node ID of the vCenter Server instance to filter subsequent endpoint requests. You can use the node ID in a function that retrieves the endpoint URL of a service on a vCenter Server instance. For information about implementing such a function, see "Retrieve Service Endpoints on vCenter Server Instances," on page 21.

Java Example of Retrieving a vCenter Server ID by Using the Lookup Service

This example is based on the in the `LookupServiceHelper.java` sample file. The file is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/com/vmware/vcloud/suite/samples/common/`

```
  ...

  getMgmtNodeId(String targetNodeName)
  {
    // 1 – List the vCenter Server instances.
    List<LookupServiceRegistrationInfo> serviceInfos =
            lookupServiceUrls("com.vmware.cis",
                              "vcenterserver",
                              "vmomi",
                              "com.vmware.vim");

   // 2 – Find the matching node name and save the ID.
    for (LookupServiceRegistrationInfo serviceInfo : serviceInfos) {
      for (LookupServiceRegistrationAttribute serviceAtttr : serviceInfo.getServiceAttributes())
{
        if ("com.vmware.vim.vcenter.instanceName".equals(serviceAttr.getKey())) {
          if (serviceAttr.getValue().equals(targetNodeName)) {
            return serviceInfo.getNodeId();
          }
        }
      }
```

```
    }
  }

  ...
```

Retrieve a vCloud Suite Endpoint on a vCenter Server Instance

Through the vCloud Suite Endpoint, you can access other vCloud Suite services that run on vCenter Server, such as Content Library and Tagging. To use a vCloud Suite service, you must retrieve the vCloud Suite Endpoint.

Prerequisites

- Establish a connection with the Lookup Service.
- Retrieve a service registration object.
- Determine the node ID of the vCenter Server instance where the vCloud Suite service runs.
- Implement a function that retrieves the endpoint URL of a service on a vCenter Server instance.

Procedure

1. Invoke the function for retrieving the endpoint URL of a service on a vCenter Server instance by passing filter strings that are specific to the vCloud Suite endpoint.
2. Save the URL from the resulting single-element list.

Java Example of Retrieve a vCloud Suite Endpoint on a vCenter Server Instance

This example is based on the in the `LookupServiceHelper.java` sample file. The file is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/com/vmware/vcloud/suite/samples/common/`

```
...

//1 – Determine management node ID.
String targetNodeId = getMgmtNodeId(targetNodeFqdn);

//2 – List filtered registration info.
List<LookupServiceRegistrationInfo> results =
                  lookupSingleServiceUrl(“com.vmware.cis”,
                                         “cs.vapi”,
                                         “vapi.json.https.public”,
                                         “com.vmware.vapi.endpoint”,
                                         targetNodeId);

//3 – Extract endpoint URL from registration info.
LookupServiceRegistrationInfo registrationInfo = results.get(0);
LookupServiceRegistrationEndpoint serviceEndpoint =
registrationInfo.getServiceEndpoints().get(0);
String ssoUrl = serviceEndpoint.getUrl();

...
```

Authentication Mechanisms

4

To perform operations on services in the vSphere environment, you must create an authenticated connection to the services that you want to use. With the vCloud Suite SDK for Java you can authenticate and access vCloud Suite services.

Client applications can choose from two supported authentication patterns for accessing services in the virtual environment.

For better security, client applications can request a security token to authenticate connections with the vCloud Suite services.

To invoke operations on services, client applications must create a security context. The security context represents the client authentication. You can achieve authentication by using one of the following mechanisms.

Token-Based Authentication

In the vSphere environment, client applications can authenticate by using the vCenter Single Sign-On component on thePlatform Services Controller. vCenter Single Sign-On includes the Security Token Service (STS) that issues security tokens. The token must comply with the Security Assertion Markup Language (SAML) specification, which defines an XML-based encoding for communicating authentication data.

vCenter Single Sign-On supports two types of security tokens: bearer token and Holder-of-Key (HoK) token. To acquire a SAML token, client applications must issue a token request to vCenter Single Sign-On.

Client applications can present a SAML token to the vCloud Suite Endpoint in exchange for a session identifier with which they can perform a series of authenticated operations.

To retrieve a session ID for the vSphere Web Services endpoint, you provide the SAML token to the vSphere Web services endpoint. For more information about creating an authenticated session to access the vSphere Web Services, see the *vSphere Web Services SDK Programming Guide* documentation.

Password-Based Authentication

To authenticate without a SAML token, you connect to the vCloud Suite Endpoint with vCenter Single Sign-On user credentials and obtain a session identifier (ID). The user account credentials are validated by the vCloud Suite Endpoint, and must be present in the vCenter Single Sign-On identity store. The session ID is valid only for the service endpoint that you want to access and that issues the session ID.

This chapter includes the following topics:

Retrieve a SAML Token

The vCenter Single Sign-On service provides authentication mechanisms for securing the operations that your client application performs in the virtual environment. Client applications use SAML security tokens for authentication.

Client applications use the vCenter Single Sign-On service to retrieve SAML tokens. For more information about how to acquire a SAML security token, see the *vCenter Single Sign-On Programming Guide* documentation.

Prerequisites

Verify that you have the vCenter Single Sign-On URL. You can use the Lookup Service on the Platform Services Controller to obtain the endpoint URL. For information about retrieving service endpoints, see Chapter 3, "Retrieving Service Endpoints," on page 17.

Procedure

1 Create a connection object to communicate with the vCenter Single Sign-On service.

  Pass the vCenter Single Sign-On endpoint URL, which is provided by the Lookup Service.

2 Issue a security token request by sending valid user credentials to the vCenter Single Sign-On service on the Platform Services Controller.

The vCenter Single Sign-On service returns a SAML token.

What to do next

You can present the SAML token to the vCloud Suite Endpoint or other endpoints, such as the vSphere Web Services Endpoint. The endpoint returns a session ID and establishes a persistent session with that endpoint. Each endpoint that you connect to uses your SAML token to create its own session.

Java Example of Retrieving a SAML Token

The example is based on the code in the `ConnectionWorkflow` Java class from the `com.vmware.vcloud.suite.samples.workflow` package. The source file is located in the following vCloud Suite SDK for Java directory: `VMware-vCloud-Suite-SDK-Java/client/samples/java`.

```
import com.vmware.vcloud.suite.samples.common.SSOConnection;
import com.vmware.vapi.saml.SamlToken;

...

String ssoUrl = lookupService.findSsoUrl();

// Create a connection objects to communicate with the vCenter Single Sign-On service.
SSOConnection ssoConnection = getSSOConnection(ssoUrl);
```

```
// Send user credentials to acquire a bearer SAML token.
ssoConnection.login(ssoUsername, ssoPassword);
SamlToken samlBearerToken = ssoConnection.getSamlBearerToken();
```

Create a vCloud Suite Session with a SAML Token

To establish a vCloud Suite session, you create a connection to the vCloud Suite Endpoint and then you create a session for the connection.

Prerequisites

- Retrieve the vCloud Suite Endpoint URL from the Lookup Service.
- Obtain a SAML token from the vCenter Single Sign-On service.

Procedure

1. Create a connection by specifying the vCloud Suite Endpoint URL and the message protocol to be used for the connection.

   NOTE To transmit your requests securely, use **`https`** for the vCloud Suite Endpoint URL.

2. Create a stub configuration instance and set the security context to be used.

   The security context object contains the SAML token retrieved from the vCenter Single Sign-On service. Optionally, the security context might contain the private key of the client application.

3. Create a `Session` stub that uses the stub configuration instance.
4. Create an authenticated session to the vCloud Suite Endpoint by using the `Session` stub.

   The operation returns a session identifier.

5. Create a security context instance and add the session ID to it.
6. Update the stub configuration instance with the session security context.

What to do next

You can use the authenticated session to access vCloud Suite services. For more information about creating stubs to the vCloud Suite services, see Chapter 5, "Accessing vCloud Suite Services," on page 31.

Java Example of Creating a vCloud Session with a SAML Token

This example is based on the code in the `ConnectionWorkflow` Java class from the `com.vmware.vcloud.suite.samples.workflow` package. The source file is located in the following vCloud Suite SDK for Java directory: `VMware-vCloud-Suite-SDK-Java/client/samples/java`.

```
import com.vmware.cis.tagging.Tag;
import com.vmware.vapi.bindings.StubConfiguration;
import com.vmware.vapi.bindings.StubFactory;
import com.vmware.vapi.cis.authn.ProtocolFactory;
import com.vmware.vapi.protocol.ProtocolConnection;
import com.vmware.vapi.cis.authn.SecurityContextFactory;
import com.vmware.vapi.core.ExecutionContext.SecurityContext;

// Create the protocol connection object to access the vCloud Suite endpoint.
ProtocolFactory protocolFactory = new ProtocolFactory();
ProtocolConnection protocolConnection = protocolFactory.getConnection("http", vCloudSuiteUrl,
                                        SslUtil.createTrustStoreForServer(vCloudSuiteUrl));
```

```
// Create a stub factory.
StubFactory myStubFactory = new StubFactory(protocolConnection.getApiProvider());

// Create the stub configuration instance and
// add a SAML token security context to the instance.
StubConfiguration myStubConfiguration = new StubConfiguration();
SamlTokenSecurityContext tokenContext = SecurityContextFactory.createSamlSecurityContext
                                          (ssoConnection.getSamlBearerToken(), null);
myStubConfiguration.setSecurityContext(tokenContext);

// Create a session stub that uses the SAML token security context.
Session mySession = myStubFactory.createStub(Session.class, stubConfiguration);

// Create an authenticated session with the vCloud Suite service.
char[] sessionId = mySession.create();

// Create a session security context and
// set the stub configuration to use the session ID.
SessionSecurityContext sessionContext =
                               SecurityContextFactory.createSessionSecurityContext(sessionId);
myStubConfiguration.setSecurityContext(sessionContext);
Tag taggingServiceStub = myStubFactory.createStub(Tag.class, myStubConfiguration);
```

Create a vCloud Suite Session with User Credentials

With the vCloud Suite SDK for Java , you can create authenticated sessions by using only user credentials.

You connect to the vCloud Suite Endpoint by using a user name and password known to the vCenter Single Sign-On service. The vCloud Suite uses your credentials to authenticate with the vCenter Single Sign-On Service and obtain a SAML token.

Prerequisites

- Retrieve the vCloud Suite Endpoint URL from the Lookup Service.
- Verify that you have valid user credentials for the vCenter Single Sign-On identity store.

Procedure

1. Create a connection stub by specifying the vCloud Suite Endpoint URL and the message protocol to be used for the connection.
2. Create a stub configuration instance and set the specific security context to be used.

   The security context object uses the user name and password that are used for authenticating to the vCenter Single Sign-On service.
3. Create a `Session` stub that uses the stub configuration instance.
4. Call the create operation on the `Session` stub to create an authenticated session to the vCloud Suite Endpoint.

   The operation returns a session identifier.
5. Create a security context instance and add the session ID to it.
6. Update the stub configuration instance with the session security context.

What to do next

You can use the authenticated session to access vCloud Suite services. For more information about creating stubs to the vCloud Suite services, see Chapter 5, “Accessing vCloud Suite Services,” on page 31.

Java Example of Creating a vCloud Suite Session with User Credentials

This example is based on the code in the `VapiConnectionWorkflow` Java class from the `com.vmware.vcloud.suite.samples.workflow` package. The source file is located in the following vCloud Suite SDK for Java directory: `VMware-vCloud-Suite-SDK-Java/client/samples/java`.

```
import com.vmware.vapi.bindings.StubConfiguration;;
import com.vmware.vapi.bindings.StubFactory;
import com.vmware.vapi.cis.authn.ProtocolFactory;
import com.vmware.vapi.protocol.ProtocolConnection;
import com.vmware.vapi.cis.authn.SecurityContextFactory;
import com.vmware.vapi.cis.authn.UserPassSecurityContext;
import com.vmware.vapi.security.SessionSecurityContext;

...

// Create protocol connection object to access the vCloud Suite Endpoint.

ProtocolFactory protocolFactory = new ProtocolFactory();
ProtocolConnection protocolConnection = protocolFactory.getConnection("Http", vCloudSuiteUrl,
                                            SslUtil.createTrustStoreForServer(vapiUrl));
StubFactory myStubFactory = new StubFactory(protocolConnection.getApiProvider());

// Create stub configuration instance and add the user name/password security context to it.
StubConfiguration myStubConfiguration = new StubConfiguration();
UserPassSecurityContext basicContext =
             SecurityContextFactory.createUserPassSecurityContext(ssoUsername, ssoPassword);
myStubConfiguration.setSecurityContext(basicContext);

// Create session with the user name/password security context.
Session session = stubFactory.CreateStub(Session.class, myStubConfiguration);

// Create an authenticated session to the vCloud Suite Endpoint
// by using the user name/password security context.
char[] sessionId = session.create();

// Create a security context instance and add the session ID.
myStubConfiguration = new StubConfiguration();
SessionSecurityContext sessionContext = SecurityContextFactory.createSessionSecurity(sessionId);

// Update the stub configuration to use the session security context.
myStubConfiguration.setSecurityContext(sessioncontext);
mySession = myStubFactory.createStub(Session.class, myStubConfiguration);
```

Create a Web Services Session

To develop a complex workflow, you might need to send requests to vSphere Web Services running in your virtual environment. To achieve this, you access the vSphere Web Services API by using the Web Services endpoint.

The vSphere Web Services API also supports session-based access. To establish an authenticated session, you can send the SAML token retrieved from the vCenter Single Sign-On service to a vSphere Web Service. In return you receive a session identifier that you can use to access the service. For more information about accessing Web Services, see the *vSphere Web Services SDK Programming Guide* documentation.

Prerequisites

- Retrieve the vSphere Web Services endpoint URL from the Lookup Service.
- Obtain a SAML token from the vCenter Single Sign-On service.

Procedure

1 Connect to the vSphere Web Services endpoint.

2 Send the SAML token to a specific Web service to create an authenticated session.

3 Add the retrieved session ID to the service content object.

The Service Content object gives you access to several server-side managed objects that represent vSphere services and components.

Java Example of Creating a vSphere Web Services Session

This example is based on the code in the `ServiceManager` and the `SSOConnection` Java classes from the `com.vmware.vcloud.suite.samples.common` package. The source files are located in the following vCloud Suite SDK for Java directory: `VMware-vCloud-Suite-SDK-Java/client/samples/java`.

```
import com.vmware.vim25.VimPortType;
import com.vmware.vim25.VimService;
import com.vmware.vsphere.samples.LoginByTokenSample;
import com.vmware.vim25.ServiceContent;
import com.vmware.vcloud.suite.samples.vim.helpers.VimUtil;

...

// Log in to the vSphere Web Services Endpoint and retrieve a session identifier.
SamlTokenElement tokenElement = ssoConnection.getSamlBearerTokenElement();
String sessionId = LoginByTokenSample.LoginUsingSAMLToken(tokenElement, vimUrl, null, null);

// Use the VimPortType and VimService objects from
// the vSphere Web Services API for accessing Web Services and
// retrieve the request context.
VimService vimService = new VimService();
vimPortType vimPort = vimService.getVimPort();

// Add the retrieved session ID to the request context.
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vimUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
Map<String, List<String>> headers =
            (Map<String, List<String>>) ctxt.getMessageContext.HTTP_REQUEST_HEADERS);
  if (headers == null) {
     headers = new HashMap<String, List<String>>();
  }
headers.put("Cookie", Arrays.asList(vcSessionId);
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);

// Use the session ID context when retrieving the ServiceContent object.
// The ServiceContent object gives you access to a number of
// server-side managed objects that represent vSphere services and components.
// For more information about the vSphere Web Services,
// see the vSphere Web Services SDK Programming Guide documentation.
ServiceContent serviceContent = VimUtil.getServiceContent(vimPort);
```

Accessing vCloud Suite Services 5

vCloud Suite SDK provides mechanisms for creating remote stubs to give clients access to vCloud Suite services.

The sequence of tasks you must follow to create a remote stub starts with creating a `ProtocolFactory`. You use the protocol factory object to create a `ProtocolConnection`. Connection objects provide the basis for creating stub interfaces tovCloud Suite services.

When you establish a connection to the vCloud Suite Endpoint, you can create a `StubFactory` object and a `StubConfiguration` object. With these objects, you can create the remote stub for the vCloud Suite service that you want to access.

The complete connection sequence also includes SSL truststore support and a temporary `StubConfiguration` that you use for SAML token authentication and session creation.

SSL Handshake

The vCloud Suite Endpoint (`https://host/api`) is an SSL-enabled service that requires client authentication during login. The SSL connection relies on certificate verification supported by the Java security architecture. The Java security architecture defines truststores for SSL connections. A truststore contains vCenter Single Sign-On credentials. You use a truststore to verify credentials from a vCenter Server instance.

The vCloud Suite SDK for Java includes an SSL utility sample code that supports the creation of a truststore for the HTTP connection, `com.vmware.vcloud.suite.samples.common.SslUtil`.

NOTE The vCloud Suite SDK for Java SSL utility creates an instance of the Java security certificate class `X509TrustManager`. This instance declares an override client-side method, `checkServerTrusted`, that accepts all HTTPS certificates. This method is suitable only for development environments. For a production environment, do not use the `X509TrustManager` override methods. Instead, set up a truststore for use by the default `X509TrustManager` implementation.

Access a vCloud Suite Service

To access a vCloud Suite service, you must have a valid session connection. The sequence for accessing a vCloud Suite service includes creating a protocol connection object and using it to create the service stub.

Prerequisites

Establish a connection to the vCloud Suite Endpoint URL. For more information about the authentication mechanisms that you can use, see Chapter 4, “Authentication Mechanisms,” on page 25.

Procedure

1 Create a protocol factory object.

2 Create a protocol connection object to access an API provider.

The vCloud Suite API clients use `ApiProvider` instances to invoke operations on services running in the virtual environment. To invoke an operation, you must specify the target service and operation, input parameters, and execution context.

3 Create a `StubFactory` object by using the `ApiProvider` instance.

4 Create a `StubConfiguration` instance and set the security context to be used for the service stub.

5 Create the stub for the vCloud Suite service interface by calling the create method of the `StubFactory` instance. Pass the service class and the `StubConfiguration` instance as arguments.

Java Example of Accessing a vCloud Suite Service

The example is based on the code in the `ConnectionWorkflow` Java class from the `com.vmware.vcloud.suite.samples.workflow` package. The source file is located in the following vCloud Suite SDK for Java directory: `VMware-vCloud-Suite-SDK-Java/client/samples/java`.

This example shows the steps for creating an authenticated session to the vCloud Suite Endpoint and creating the service stub for the Content Library API provider.

```
import com.vmware.cis.tagging.Tag;
import com.vmware.vapi.bindings.StubConfiguration;
import com.vmware.vapi.bindings.StubFactory;
import com.vmware.vapi.cis.authn.ProtocolFactory;
import com.vmware.vapi.cis.authn.SecurityContextFactory;
import com.vmware.vapi.protocol.ProtocolConnection;
import com.vmware.vcloud.suite.samples.common.SSOConnection;
import com.vmware.vcloud.suite.samples.common.LookupServiceHelper;

...

// Log in to vCenter Single Sign-On by using valid user credentials.
SSOConnection ssoConnection = getSSOConnection(ssoUrl);
ssoConnection.login(ssoUsername, ssoPassword);

// Create a protocol factory.
ProtocolFactory protocolFactory = new ProtocolFactory();

// Retrieve the vCloud Suite Endpoint URL.
LookupServiceHelper lookupService = new LookupServiceHelper(lookupServiceUrl);
String vCenterServerId = lookupService.getMgmtNodeId(vCenterServerName);
String vCloudSuiteUrl = lookupService.findVapiUrl(vCenterServerId);

// Create the protocol connection.
ProtocolConnection protocolConnection = protocolFactory.getConnection("https",
                                                                      vCloudSuiteUrl,

SslUtil.createTrustStoreForServer(vCloudSuiteUrl));

// Create a StubFactory object.
StubFactory stubFactory = new StubFactory(protocolConnection.getApiProvider());

// Create the stub configuration instance and add a SAML security context to the instance.
```

```
StubConfiguration stubConfiguration = new StubConfiguration();
stubConfiguration.setSecurityContext(SecurityContextFactory.
         createSamlSecurityContext(ssoConnection.getSamlBearerToken(), null));

// Create a session and obtain a session identifier for the vCloud Suite Endpoint.
Session session = stubFactory.createStub(Session.class,    stubConfiguration);
char[] sessionId = session.create();

// Update the StubConfiguration instance with the session identifier.
stubConfiguration.setSecurityContext(SecurityContextFactory.createSessionSecurityContext(sessionI
d));

// Create service stubs for the Content Library service.
Library libraryService = stubFactory.createStub(Library.class,    stubConfiguration);

// Invoke an operation of the Content Library service.
List<String> listContentLibraries = libraryService.list();
```

Content Library Service 6

The Content Library Service provides means for managing content libraries in the context of a single or multiple vCenter Server instances deployed in your virtual environment. You can use the vCloud Suite APIs to access the Content Library Service through the vCloud Suite Endpoint.

Administrators can use content libraries to share VM templates, vApp templates, and other types of files across vCenter Server instances in the virtual environment. Sharing templates across your virtual environment promotes consistency, compliance, efficiency, and automation in deploying workloads at scale.

Content Library Overview

A content library instance represents a container for a set of library items. A content library item instance represents the logical object stored in the content library, which might be one or more usable files.

- Content Library Storage on page 36

  When you create a local library, you can store its contents on a datastore in your vCloud Suite inventory or on the local file system for thevCenter Server .

Content Library Types

You can create two types of libraries, local and subscribed.

- You can create a local library as the source for content you want to save or share. Create the local library on a single vCenter Server instance. You can add items to a local library or remove them. You can publish a local library and as a result this content library service endpoint can be accessed by other vCenter Server instances in your virtual environment. When you publish a library, you can configure the authentication method, which a subscribed library must use to authenticate to it.
- You can create a subscribed library and populate its content by synchronizing to a local library. A subscribed library contains copies of the local library files or just the metadata of the library items. The local library can be located on the same vCenter Server instance as the subscribed library, or the subscribed library can reference a local library on a different vCenter Server instance. You cannot add library items to a subscribed library. You can only add items to the source library. After synchronization, both libraries will contain the same items.

Content Library Items

Library items are VM templates, vApp templates, or other VMware objects that can be contained in a content library. In the case of VMs and vApps, there are several files, such as log files, disk files, memory files, and snapshot files that are part of a single library item. You can create library items in a specific local library or remove items from a local library. You can also upload files to an item in a local library so that the libraries subscribed to it can download the files to their local file system.

For information about the tasks that you can perform by using the content library service, see “Content Libraries,” on page 38.

Content Library Storage

When you create a local library, you can store its contents on a datastore in your vCloud Suite inventory or on the local file system for thevCenter Server .

You can use NFS or CIFS for local storage. For example, if you use a vCenter Server instance that runs on a Windows machine, you can pass a URI pointing to the CIFS shared storage mapped on the Windows machine. If you use a vCenter Server Appliance instance, you can enter the path to the NFS storage that is mounted on the Linux file system in the appliance.

Java Example of Storing Library Content on a Datastore

This example is based on the code in the `ContentLibraryWorkflow.java` sample file located in the following vCloud Suite SDK for Java directory: `/client/samples/java/com/vmware/vcloud/suite/samples/interop`.

```
...
import com.vmware.content.library.StorageBacking;
import com.vmware.content.library.StorageBacking.Type;

// Create a StorageBacking instance of datastore type.

     StorageBacking libraryBacking = new StorageBacking();
     libraryBacking.setType(Type.DATASTORE);

/*
 * Pass the value of the datastore ManagedObjectReference.
```

```
 * See the vSphere Web Services SDK Programming Guide
 * and the vSphere Web Services SDK samples. In addition, the vCloud Suite SDK for Java
provides
 * the VimUtil utility class in the com.vmware.vcloud.suite.samples.vim.helpers package.
 * You can use the utility to retrieve the ManagedObjectReference
 * of the datastore entity.
 */
     libraryBacking.setDatastoreId("datastore-123");

// Create a LibraryModel that represents a local library backed on a datastore.

      LibraryModel libraryModel = new LibraryModel();
      libraryModel.setName("AcmeLibrary");
      libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
```

Querying Content Libraries

You can create queries to find libraries that match your criteria. You can also retrieve a list of all libraries or only the libraries of a specific type.

- Listing All Content Libraries on page 37

  You can retrieve a list of all content library IDs in your virtual environment, regardless of their type, by using the `Library` service.

- Listing Content Libraries of a Specific Type on page 38

  You can use the vCloud Suite API to retrieve content libraries of a specific type. For example, you can list only the local libraries in your virtual environment.

- List Content Libraries by Using Specific Search Criteria on page 38

  You can filter the list of content libraries and retrieve only the libraries that match your specific criteria. For example, you might want to publish all local libraries with a specific name.

Listing All Content Libraries

You can retrieve a list of all content library IDs in your virtual environment, regardless of their type, by using the `Library` service.

You can use the `list()` function to retrieve all local and subscribed libraries in your system.

Java Example of Retrieving a List of All Content Libraries

The example is based on the code in the `ContentLibraryWorkflow.java` sample file. The sample resource is located in the following vCloud Suite SDK for Java directory, `client/samples/java/com/vmware/vcloud/suite/samples/interop`.

```
...
import com.vmware.content.Library;
import com.vmware.vcloud.suite.samples.common.ServiceManager;

// Access the Library Service.
   Library libraryService = serviceManager.getVapiService(Library.class);

// List all content libraries.
   List<String> allLibraries = libraryService.list();
```

```
System.out.println("List of all library identifiers: /n");
for (String cl : allLibraries) {
      System.out.println(cl);
}
```

Listing Content Libraries of a Specific Type

You can use the vCloud Suite API to retrieve content libraries of a specific type. For example, you can list only the local libraries in your virtual environment.

If you want to retrieve only a list of the local libraries, you must retrieve the `LocalLibrary` service and use the `list` function on the `LocalLibrary` service. To list only subscribed libraries, you must retrieve the `SubscribedLibrary` service and call the `list` function on the `SubscribedLibrary` service.

List Content Libraries by Using Specific Search Criteria

You can filter the list of content libraries and retrieve only the libraries that match your specific criteria. For example, you might want to publish all local libraries with a specific name.

Prerequisites

Verify that you have access to the `Library` service.

Procedure

1 Create a `FindSpec` instance and specify your search criteria.

2 Call the `find` function on the `Library` service.

All content libraries that match your search criteria are listed.

Java Example of Retrieving a List of All Local Libraries with a Specific Name

This example retrieves a list of all local libraries with the name **`AcmeLibrary`** that exist in your virtual environment.

```
import com.vmware.content.LibraryModel;
import com.vmware.content.LibraryModel.LibraryType;
import com.vmware.content.LibraryTypes.FindSpec;

// Create a FindSpec instance to set your search criteria.
   FindSpec findSpec = new FindSpec();

// Filter the local content libraries by using a library name.
   findSpec.setName("AcmeLibrary");
   findSpec.setType(LibraryType.LOCAL);
   List<String> ids = libraryService.find(findSpec);
```

Content Libraries

The Content Library API provides services that allow you to create and manage content libraries programmatically. You can create a local library and publish it for the entire virtual environment. You can also subscribe to use the contents of a local library and enable automatic synchronization to ensure that you have the latest content.

- Create a Local Content Library on page 39

  You can create a local content library programmatically by using the vCloud Suite API. The API allows you to populate the content library with VM and vApp templates. You can use these templates to deploy virtual machines or vApps in your virtual environment.

- Publish an Existing Content Library on page 40

  To make the library content available for other vCenter Server instances across the vCloud Suite environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that already exists in your vCloud Suite environment.

- Publish a Library at the Time of Creation on page 41

  You can publish a local library at the time of creation to enable other libraries to subscribe and use the library content.

- Subscribe to a Content Library on page 41

  You can subscribe to public content libraries. The source objects for a public content library can be: a library created on a vCenter Server 6.0 instance, a catalog created on a vCloud Director 5.5 instance, or a third-party library. When you subscribe to a library, you must specify the backing storage for the library content. You must also provide the correct user name and password if the library requires basic authentication.

- Synchronize a Subscribed Content Library on page 43

  When you subscribe to a published library, you can configure the settings for downloading and updating the library content.

- Editing the Settings of a Content Library on page 44

  You can update the settings of content library types in your virtual environment by using the vCloud Suite API.

- Removing the Content of a Subscribed Library on page 44

  You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

- Delete a Content Library on page 45

  When you no longer need a content library, you can invoke the `delete` method on either the `LocalLibrary` or the `SubscribedLibrary` service depending on the library type.

Create a Local Content Library

You can create a local content library programmatically by using the vCloud Suite API. The API allows you to populate the content library with VM and vApp templates. You can use these templates to deploy virtual machines or vApps in your virtual environment.

Procedure

1 Create a `StorageBacking` instance and define the storage location.

2 Create a `LibraryModel` instance and set the properties of the new local library.

3 Access the `LocalLibrary` object which is part of the vCloud Suite API service interfaces.

4 Call the create function on the `LocalLibrary` object and pass the `LibraryModel` as a parameter.

Java Example of Creating a Local Library

This example is based on the code in the `ContentLibraryWorkflow.java` sample file. The sample resource is located in the following vCloud Suite SDK for Java directory, `client/samples/java/com/vmware/vcloud/suite/samples/interop`.

```
...
import com.vmware.content.LibraryModel;
import com.vmware.content.library.StorageBacking;
import com.vmware.content.library.StorageBacking.Type;
```

```
// Create a StorageBacking instance to back the library content on the local file system.
     StorageBacking libraryBacking = new StorageBacking();
     libraryBacking.setType(Type.OTHER);
     libraryBacking.setStorageUri(URI.create("file:///tmp"));
     libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));

// Create a LibraryModel that represents a local library.
     LibraryModel libraryModel = new LibraryModel();
     libraryModel.setType(LibraryModel.LibraryType.LOCAL);
     libraryModel.setName("AcmeLibrary");

// Access the LocalLibrary service by using the  endpoint.
     LocalLibrary localLibraryService = serviceManager.getVapiService(LocalLibrary.class);

// Call the create method of the LocalLibrary service passing as an
// argument the LibraryModel instance.
     String libraryId = localLibraryService.create(UUID.randomUUID().toString(), libraryMod
```

Publish an Existing Content Library

To make the library content available for other vCenter Server instances across the vCloud Suite environment, you must publish the library. Depending on your workflow, select a method for publishing the local library. You can publish a local library that already exists in your vCloud Suite environment.

Procedure

1 Retrieve a reference to the `LocalLibrary` service.

2 Retrieve an existing local library by using the library ID.

3 Create a `PublishInfo` instance to define how the library is published.

4 Specify the authentication method to be used by a subscribed library to authenticate to the local library. You can enable either basic authentication or no authentication. Basic authentication requires a user name and password.

5 (Optional) If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

IMPORTANT Use the predefined user name **vcsp** or leave the user name undefined. You must set only a password to protect the library.

---

6 Specify that the library is published.

7 Use the retrieved local library to configure it with the `PublishInfo` instance.

8 Update the properties of the `LibraryModel` object returned for the local library.

Java Example of Publishing an Existing Content Library

This example is based on the code in the `ContentLibraryWorkflow.java` sample file. The sample resource is located in the following vCloud Suite SDK for Java directory `client/samples/java/com/vmware/vcloud/suite/samples/interop`.

```
...
import com.vmware.content.LibraryModel;
import com.vmware.content.LocalLibrary;
import com.vmware.content.library.PublishInfo;
import com.vmware.content.library.StorageBacking;
...
```

```
// Access the LocalLibrary service.
      LocalLibrary localLibraryService = serviceManager.getVapiService(LocalLibrary.class);

// Retrieve an existing local library.
      LibraryModel libraryModel = localLibraryService.get(libraryId);
      PublishInfo publishInfo = new PublishInfo();

// Configure how the local library is published by using BASIC authentication.
      publishInfo.setUserName("vcsp");
      publishInfo.setPassword("password".toCharArray());
      publishInfo.setAuthenticationMethod(AuthenticationMethod.BASIC);

// Set the local library to published and update the library instance.
      publishInfo.setPublished(true);
      libraryModel.setPublishInfo(publishInfo);
      localLibraryService.update(libraryModel.getId(), libraryModel);
```

Publish a Library at the Time of Creation

You can publish a local library at the time of creation to enable other libraries to subscribe and use the library content.

Procedure

1 Retrieve the `LocalLibrary` service.

2 Create a `PublishInfo` instance to define how the library is published.

3 Specify the authentication method to be used by a subscribed library to authenticate to the local library.

You can enable either basic authentication or no authentication on the library. Basic authentication requires a user name and password.

4 (Optional) If you publish the library with basic authentication, you must specify a user name and password for the `PublishInfo` instance, which must be used for authentication.

IMPORTANT Use the predefined user name **vcsp** or leave the user name undefined. You must set only a password to protect the library.

5 Create a `LibraryModel` instance and configure the instance.

6 Set the library type to local and use the configured `PublishInfo` instance to set the library to published.

7 Define where the content of the local library is stored by using the `StorageBacking` class.

8 Create a published local library.

Subscribe to a Content Library

You can subscribe to public content libraries. The source objects for a public content library can be: a library created on a vCenter Server 6.0 instance, a catalog created on a vCloud Director 5.5 instance, or a third-party library. When you subscribe to a library, you must specify the backing storage for the library content. You must also provide the correct user name and password if the library requires basic authentication.

NOTE If you subscribe to libraries created with basic authentication on a vCenter Serverinstance, make sure that you pass **vcsp** as an argument for the user name.

Procedure

1 Create a `StorageBacking` instance to define the location where the content of the subscribed library is stored.

2 Create a `SubscriptionInfo` instance to define the subscription behavior of the library.

   a Provide the mechanism to be used by the subscribed library to authenticate to the published library.

   You can choose between no authentication and basic authentication depending on the settings of the published library you subscribe to. If the library is published with basic authentication, you must set basic authentication in the `SubscriptionInfo` instance. Set the user name and the password of the `SubscriptionInfo` instance to match the credentials of the published library.

   b Provide the URL to the endpoint where the metadata of the published library is stored.

   c Define the synchronization mechanism of the subscribed library.

   You can choose between two synchronization modes: automatic and on demand. If you enable automatic synchronization for a subscribed library, both the content and the metadata are synchronized with the published library. To save storage space, you can synchronize the subscribed library on demand and update only the metadata for the published library content.

   d Set the thumbprint that is used for validating the certificate of the published library.

3 Create a `LibraryModel` instance and set the library type to subscribed (`LibraryModel.LibraryType.SUBSCRIBED`).

4 Access the `SubscribedLibrary` service and create the subscribed library instance.

Java Example of Subscribing to a Published Library

This example is based on the code in the `ContentLibraryWorkflow.java` sample file. The sample resource is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/com/vmware/vcloud/suite/samples/interop` .

```
...
import com.vmware.content.LibraryModel;
import com.vmware.content.SubscribedLibrary;
import com.vmware.content.library.SubscriptionInfo;
import com.vmware.content.library.StorageBacking;
...

// Create a StorageBacking instance to store
// the contents of the subscribed library on the local file system.
       StorageBacking libraryBacking = new StorageBacking();
       libraryBacking.setType(StorageBacking.Type.OTHER);
       libraryBacking.setStorageUri(URI.create("/mnt/nfs/cls-root"));

// Create a new SubscriptionInfo object to define the subscription
// behavior of the library.
       SubscriptionInfo subscriptionInfo = new SubscriptionInfo();
       subscriptionInfo.setAuthenticationMethod
         (com.vmware.content.library.SubscriptionInfo.AuthenticationMethod.BASIC);
       subscriptionInfo.setUserName("libraryUser");
       subscriptionInfo.setPassword("password".toCharArray());

subscriptionInfo.setSubscriptionUrl(URI.create("https://www.acmecompary.com/library_inventory/lib
.json"));
```

```
// Specify that the content of the subscribed library will be downloaded immediately.
       subscriptionInfo.setAutomaticSyncEnabled(true);

// Set an SHA-1 hash of the SSL certificate of the remote endpoint.
       subscriptionInfo.setSslThumbprint("9B:00:3F:C4:4E:B1:F3:F9:0D:70:47:48:E7:0B:D1:A7:0E:DE:
60:A5");

// Create a new LibraryModel object for the subscribed library.
       LibraryModel libraryModel = new LibraryModel();
       libraryModel.setType(LibraryModel.LibraryType.SUBSCRIBED);
       libraryModel.setName("SubscrLibrary");

// Attach the storage backing and the subscription info to the library model.
       libraryModel.setStorageBackings(Collections.singletonList(libraryBacking));
       libraryModel.setSubscriptionInfo(subscriptionInfo);

// Create the new subscribed library.
       String clientToken = UUID.randomUUID().toString();
       SubscribedLibrary subscribedLibService =
serviceManager.getVapiService(SubscribedLibrary.class);
       String subscribedLibId = subscribedLibService.create(clientToken, libraryModel);
```

Synchronize a Subscribed Content Library

When you subscribe to a published library, you can configure the settings for downloading and updating the library content.

- You can enable automatic synchronization of the subscribed library and download a copy of the content of the local library immediately.
- You can save storage space and download only the metadata for the items that are part of the local library.

To ensure that your subscribed library contains the most recent published library content, you can force a synchronization task.

Procedure

1 Access the `SubscribedLibrary` vCloud Suite service.

2 Retrieve the subscribed library ID from the `SubscribedLibrary` service.

3 Force the synchronization of the subscribed library.

The synchronization operation depends on the update settings of the subscribed library. If the subscribed library is configured to update only on demand, only the metadata of the library items will be synchronized.

Editing the Settings of a Content Library

You can update the settings of content library types in your virtual environment by using the vCloud Suite API.

Table 6-1. Options for Updating a Content Library

Content Library Types	Option

Local content library	You can change the settings of a local library before calling the update function on the `LocalLibrary` object:
	■ Before a library is published, you can edit the following settings:
	  ■ The name of a local library that is retrieved by using the `LocalLibrary` object
	  ■ The human-readable description of a local library retrieved by using the `LocalLibrary` object
	■ After a library is published, you must first retrieve the `PublishInfo` instance of the published library you want. You can use the instance to configure the following settings.
	  ■ Unpublish the local library.
	  ■ Change the authentication method of the library.
	  ■ Change the password that must be used for authentication.

Subscribed content library	You can edit the settings of a subscribed library if you retrieve the `SubscriptionInfo` instance associated with it. To apply the changes, you must update the library by using the `SubscribedLibrary` object. You can configure the following settings: ■ The authentication method required by the local library ■ The user name and password of the subscribed library for authentication to the local library ■ The method for synchronizing the metadata and the content of the subscribed library ■ The thumbprint used for validating the SSL certificate of the local library

Removing the Content of a Subscribed Library

You can free storage space in your virtual environment by removing the subscribed library content that you no longer need.

You can create a subscribed library with the option to download the library content on demand. As a result, only the metadata for the library items is stored in the associated with the subscribed library storage. When you want to deploy a virtual machine from a VM temple in the subscribed library, you must synchronize the subscribed library to download the entire published library content. When you no longer need the VM template, you can call the `evict` function on the `SubscribedLibrary` service. You must provide the subscribed library ID to this function. As a result, the subscribed library content that is cached on the backing storage is deleted.

If the subscribed library is not configured to synchronize on demand, an exception is thrown. In this case the subscribed library always attempts to have the most recent published library content.

Delete a Content Library

When you no longer need a content library, you can invoke the `delete` method on either the `LocalLibrary` or the `SubscribedLibrary` service depending on the library type.

Procedure

1 Access the `SubscribedLibrary` or the `LocalLibrary` service by using the vCloud Suite Endpoint.

2 Retrieve the library ID you want to delete.

3 Call the `delete` function on the library service and pass the library ID as argument.

All library items cashed on the storage backing are removed asynchronously. If this operation fails, you must manually remove the content of the library.

Library Items

A library item groups multiple files within one logical unit. You can perform various tasks with the items in a content library.

You can upload files to a library item in a local library and update existing items. You can download the content of a library item from a subscribed library and use the item, for example, to deploy a virtual machine. You can remove the content of a library item from a subscribed library to free storage space and keep only the metadata of the library item. When you no longer need local library items, you can delete them and they are removed from the subscribed library when a synchronization task is completed.

You can create a library item from a specific item type, for example **`.ovf`**. The Content Library service must support the library item type to handle the item correctly. If no support is provided for a specified type, the Content Library service handles the library item in the default way, without adding metadata to the library item or guiding the upload process.

- Create an Empty Library Item on page 46

  You can create as many library items as needed and associate them with a local content library.

- Querying Library Items on page 47

  You can perform numerous query operations on library items.

- Edit the Settings of a Library Item on page 47

  You can edit the name, description, and type of a library item.

- Upload a File from a Local System to a Library Item on page 48

  You can upload different types of files from a local system to a library item that you want to use in the vCloud Suite environment.

- Upload a File from a URL to a Library Item on page 50

  You can upload different types of files from a local system to a library item that you want to use in the vCloud Suiteenvironment.

- Download Files to a Local System from a Library Item on page 51

  You might want to download files to a local system from a library item and then make changes to the files before you use them.

- Synchronizing a Library Item in a Subscribed Content Library on page 53

  The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library.

- Removing the Content of a Library Item on page 53
  You can remove the content from a library item to free space on your storage.
- Deleting a Library Item on page 53
  You can remove a library item from a local library when you no longer need it.

Create an Empty Library Item

You can create as many library items as needed and associate them with a local content library.

Procedure

1 Access the `Item` service by using the vCloud Suite endpoint.

2 Instantiate the `ItemModel` class.

3 Define the settings of the new library item.

4 Associate the library item with an existing local library.

5 Invoke the `create` function on the `Item` object to pass the library item specification and the unique client token.

What to do next

Upload content to the new library item. See "Upload a File from a Local System to a Library Item," on page 48 and "Upload a File from a URL to a Library Item," on page 50.

Java Example of Creating a Library Item

This example shows how to create an empty library item that stores an ISO image file.

```
...
import java.util.UUID;
import com.vmware.content.library.Item;
import com.vmware.content.library.ItemModel;
...

// Create an instance of the ItemModel class and specify the item settings.
ItemModel libItemSpec = new ItemModel();
libItemSpec.setName("ESXi ISO image");
libItemSpec.setDescription("ISO image with the latest security patches for ESXi 5.5 as of
2/3/2015");
libItemSpec.setType("iso");

// Associate the item with an existing content library.
libItemSpec.setLibraryId("<content_library_ID>");

// Create the new Item instance, using the specified model.
Item libItemService = serviceManager.getVapiService(Item.class);
String itemID = UUID.randomUUID().toString();
String newItem = libItemService.create(itemID, libItemSpec);
```

Querying Library Items

You can perform numerous query operations on library items.

You can retrieve a list of all items in a library, retrieve a library item that has a specific type or name, and find a library item that is not cached on the disk. You can then update the library item content from the subscribed library.

List Library Items

You can use the `list` method of the `Item` object to retrieve a list of all items in a particular library.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1 Retrieve the ID of the content library whose items you want to list.

2 List the items of the specific library.

3 Retrieve a list of the files that belong to a library item.

You can see an example query operation in the code example for "Edit the Settings of a Library Item," on page 47. The beginning of the example lists the items of a published library and prints a list with the names and size of each file in the listed items.

List Library Items That Match Specific Criteria

You can filter the items contained in a library and retrieve only the items matching specific criteria. For example, you might want to remove or update only specific items in a library.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1 Create an instance in the `FindSpec` class.

2 Specify the filter properties by using the `FindSpec` instance.

3 List the items matching the specified filter.

A list of items matching the filter criteria is created as a result.

Edit the Settings of a Library Item

You can edit the name, description, and type of a library item.

Prerequisites

Verify that you have access to the `Item` service.

Procedure

1 Retrieve the item that you want to update.

2 Create an `ItemModel` instance.

3 Change the human-readable name and description of the library item.

4 Update the library item with the configured item model.

Java Example of Changing the Settings for a Library Item

This example shows how to find an item by using the item name and then how to change the name and description of the retrieved item.

```
...
import com.vmware.content.library.Item;
import com.vmware.content.library.ItemModel;
...

// List the items in a published library
     Item libItemService = serviceManager.getVapiService(Item.class);
     List<String> itemIds = libItemService.list(libraryId.getId());
     for (String itemId : itemIds) {
         ItemModel singleItem = libItemService.get(itemId);

// List the files uploaded to each library item and print their names and size
         com.vmware.content.library.item.File itemFilesService =
                serviceManager.getVapiService(com.vmware.content.library.item.File.class);
         List<com.vmware.content.library.item.FileTypes.Info> fileInfos =
                itemFilesService.list(itemId);
             for (com.vmware.content.library.item.FileTypes.Info singleFile : fileInfos) {
                 System.out.println("Library item with name "+ singleFile.getName() + " has size
                                    " + singleFile.getSize());
             }

// Change the name and description of the library item with the specified name
               if (singleItem.getName().equals("simpleVmTemplate")) {
                    ItemModel libItemUpdated = new ItemModel();
                    libItemUpdated.setName("newItemName");
                    libItemUpdated.setDescription("Description of the newItemName");
                    libItemService.update(singleItem.getId(), libItemUpdated);
                }
        }
```

Upload a File from a Local System to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vCloud Suite environment.

Prerequisites

- Create an empty library item. See "Create an Empty Library Item," on page 46.
- Verify that you have access to the `UpdateSession` and `File` services.

Procedure

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create an `AddSpec` instance to describe the upload method and other properties of the file to be uploaded.
4. Create the request for changing the item by using the `File` service.
5. Upload the file that is on the local system.
6. Complete and delete the update session to apply the changes to the library item.

Java Example of Uploading Files to a Library Item from a Local System

This example shows how to upload an ISO image file from a local system to a library item.

```
...
import com.vmware.content.library.item.UpdateSession;
import com.vmware.content.library.item.UpdateSessionModel;
import com.vmware.content.library.item.updatesession.File;
import com.vmware.content.library.item.updatesession.FileTypes;
import com.vmware.vcloud.suite.samples.common;
...

// Access the com.vmware.content.library.item.updatesession.File.
// and the UpdateSession services by using the vCloud Suite Endpoint.
    File uploadFileService = serviceManager.getVapiService(File.class);
    UpdateSession uploadService= serviceManager.getVapiService(UpdateSession.class);

// Create an UpdateSessionModel instance to track the changes you make to the item.
    UpdateSessionModel updateSessionModel = new UpdateSessionModel();
    updateSessionModel.setLibraryItemId(newItem);

// Create a new update session.
    String clientToken = UUID.randomUUID().toString();
    String sessionId = uploadService.create(clientToken, updateSessionModel);

// Create an instance of the HttpClient class which is part of the
// com.vmware.vcloud.suite.samples.common package.
    try {
       HttpClient httpClient = new HttpClient(true);

// Create a new AddSpec instance to describe the properties of the file to be uploaded.
       FileTypes.AddSpec fileSpec = new FileTypes.AddSpec();
       fileSpec.setName("ESXi patch as of 10/2/2015");
       fileSpec.setSourceType(FileTypes.SourceType.PUSH);

// Link the ISO file specification to the update session.
       FileTypes.Info fileInfo = uploadFileService.add(sessionId, fileSpec);

// Use the HTTP library to upload the file to the library item.
       URI uploadUri = fileInfo.getUploadEndpoint().getUri();
       java.io.File file = new java.io.File("/updates/esxi/esxi_patch.iso");
       String transferUrl = uploadUri.toURL().toString();
       httpClient.upload(file, transferUrl);

// Mark the upload session as completed.
       uploadService.complete(sessionId);
    } finally {
         uploadService.delete(sessionId);
    }
```

Upload a File from a URL to a Library Item

You can upload different types of files from a local system to a library item that you want to use in the vCloud Suiteenvironment.

Prerequisites

- Create an empty library item. See "Create an Empty Library Item," on page 46.
- Verify that you have access to the `UpdateSession` and `File` services.

Procedure

1. Create an `UpdateSessionModel` instance to track the changes that you make to the library item.
2. Create an update session by using the `UpdateSession` service.
3. Create a file specification to describe the upload method and other properties of the file to be uploaded.
4. Specify the location of the file to be uploaded by creating a `TransferEndpoint` instance.
5. Add the file source endpoint to the file specification.
6. Create a request for changing the item by using the configured file specification.
7. Complete the update session to apply the changes to the library item.

Java Example of Uploading a File from a URL to a Library Item

This example shows how to upload a file from a URL to a library item. The example is based on the code in the `ContentLibraryWorkflow.java` sample file. The sample resource is located in the vCloud Suite SDK for Java directory, `client/samples/java/com/vmware/vcloud/suite/samples/interop`.

```
...
import com.vmware.content.library.item.UpdateSession;
import com.vmware.content.library.item.UpdateSessionModel;
import com.vmware.content.library.item.updatesession.File;
import com.vmware.content.library.item.updatesession.FileTypes;
...

// Create a new library item. See "Create an Empty Library Item," on page 46.

...

// Access the com.vmware.content.library.item.updatesession.File
// and the UpdateSession services by using the vCloud Suite Endpoint.
   File uploadFileService = serviceManager.getVapiService(File.class);
   UpdateSession uploadService= serviceManager.getVapiService(UpdateSession.class);

// Create an UpdateSessionModel instance to track the changes you make to the item.
        UpdateSessionModel updateSessionModel = new UpdateSessionModel();
        updateSessionModel.setLibraryItemId(newItem);

// Create a new update session.
        String clientToken = UUID.randomUUID().toString();
        String sessionId = uploadService.create(clientToken, updateSessionModel);

// Create a new AddSpec instance to describe the properties of the file to be uploaded.
         FileTypes.AddSpec fileSpec = new AddSpec();
         fileSpec.setName("ESXi patch as of 10/2/2015");
```

```
        fileSpec.setSourceType(SourceType.PULL);

// Specify the location from which the file is uploaded to the library item.
        TransferEndpoint endpoint = new TransferEndpoint();
        endpoint.setUri(URI.create("http://www.acme.com/patches_ESXi55/ESXi_patch.iso"));
        fileSpec.setSourceEndpoint(endpoint);
        uploadFileService.add(sessionId, fileSpec);

// Mark the session as completed.
        uploadService.complete(sessionId);
```

Download Files to a Local System from a Library Item

You might want to download files to a local system from a library item and then make changes to the files before you use them.

Procedure

1 Create a download session model to specify the item, which contains the file that you want to download.

2 Access the `File` service and retrieve the file that you want to export to your system within the new download session.

3 Prepare the files that you want to download and wait until the files are in the prepared state.

4 Retrieve the download endpoint URI of the files.

5 Download the files with an HTTP GET request.

6 Delete the download session after all files are downloaded.

Java Example of Downloading Files from a Library Item to Your Local System

This example is based on the code in the `OvfExporterHelper.java` sample file. The sample resource is located in the following vCloud Suite SDK for Javadirectory, `client/samples/java/com/vmware/vcloud/suite/samples/vapi/helpers`.

```
import com.vmware.content.library.item.DownloadSession;
import com.vmware.content.library.item.downloadsession.File;
import com.vmware.content.library.item.DownloadSessionModel;
...

// Access the DownloadSession service.
     DownloadSession downloadSessionService =
serviceManager.getVapiService(DownloadSession.class);

// Create a new download session model.
     DownloadSessionModel downloadSessionModel = new DownloadSessionModel();
     downloadSessionModel.setLibraryItemId(libItem.getId());
     String downloadSessionId = downloadSessionService.create(UUID.randomUUID().toString(),
downloadSessionModel);

// Access the File service and retrieve the files you want to export.
     File downloadFileService = serviceManager.getVapiService(File.class);
     List<FileTypes.Info> downloadFileInfos = downloadFileService.list(downloadSessionId);
     for (FileTypes.Info downloadFileInfo : downloadFileInfos) {
```

```
// Make sure all files are in the prepared state before you precede with the downloading
operation.
         downloadFileService.prepare(downloadSessionId, downloadFileInfo.getName(),
EndpointType.HTTPS);
         long timeOut = 360;
         Long endTime = System.currentTimeMillis() + timeOut * 1000;
         try {
              Thread.sleep(5000);
         } catch (InterruptedException e) {
                   System.out.println(e);
         }
         FileTypes.PrepareStatus expectedStatus =
com.vmware.content.library.item.downloadsession.File.PrepareStatus.PREPARED;
         downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());
         FileTypes.PrepareStatus currentStatus = downloadFileInfo.getStatus();
         if (currentStatus == expectedStatus) {

// When the files are prepared, you can retrieve the download information for each file.
            downloadFileInfo = downloadFileService.get(downloadSessionId,
downloadFileInfo.getName());
            try {
                 URI downloadUri = downloadFileInfo.getDownloadEndpoint().getUri();
                 String downloadUrl = downloadUri.toURL().toString();

// Execute an HTTP GET request and pass the download endpoints of the files.
                 HttpClient httpClient = new HttpClient(true);
                 InputStream inputStream = httpClient.downloadFile(downloadUrl);
                 String fileNameDownload = downloadFileInfo.getName();
                 File tmpDir = new java.io.File("tmp");
                 tmpDir.mkdir();
                 String fullPath = tmpDir.getAbsolutePath() +
                       System.getProperty("file.separator") + fileNameDownload;

// Copy the files to the directory on your machine.
                 Files.copy(inputStream,
Paths.get(fullPath),StandardCopyOption.REPLACE_EXISTING);
            } catch (MalformedURLException e) {
               System.out.println("Failed to download due to IOException!" + e);
               throw new RuntimeException("Failed to download due to IOException!", e);
            } catch (IOException e) {
               System.out.println("IO exception during download" + e);
               throw new RuntimeException("Failed to download due to IOException!", e);

// Delete the download session after all files are downloaded.
            } finally {
                 downloadFileService.delete(downloadSessionId);
            }
        } else {
            while (endTime > System.currentTimeMillis()) {
                downloadFileInfo = downloadFileService.get(downloadSessionId,
                             downloadFileInfo.getName());
                currentStatus = downloadFileInfo.getStatus();
                if (currentStatus == expectedStatus) {
                    return;
```

```
               } else if (currentStatus ==
com.vmware.content.library.item.downloadsession.File.PrepareStatus.ERROR) {
                   System.out.println("DownloadSession Info : " +
downloadSessionService.get(downloadSessionId));
                  throw new RuntimeException("Error while waiting for download file status to be
PREPARED...");
              }
          }
      }
   }
```

Synchronizing a Library Item in a Subscribed Content Library

The items in a subscribed library have features that are distinct from the items in a local library. Synchronizing the content and the metadata of an item in a subscribed library depends on the synchronization mechanism of the subscribed library.

Table 6-2. Options for Synchronizing a Library Item

Synchronization Type of the Subscribed Library	Description
Synchronized on demand	If the subscribed library is synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as arguments the library item ID and *true* . When you perform the task, both the item metadata and the content are synchronized. To synchronize only the metadata of the item, pass the library ID and *false* as arguments to the method.
Not synchronized on demand	If the subscribed library is not synchronized on demand, you can use the `sync` method on the `SubscribedItem` service and pass as argument the item ID. In this case, the content of the item is always synchronized and the Boolean value is ignored when the call is executed.
Synchronized automatically	If the subscribed library is synchronized automatically, you can also use the `sync` method to force the synchronization of an item. Method execution depends on whether the subscribed library is synchronized on demand.

Removing the Content of a Library Item

You can remove the content from a library item to free space on your storage.

If you create a subscribed library with the option to synchronize library content on demand, only the metadata for the library items is stored. When you want to use the items in the library, you must force synchronization on the items to download their content. When you no longer need the files in an item, you can remove the cached content of the library item and free storage space. To achieve this task use the `evict` function of the `SubscribedItem` object.

Deleting a Library Item

You can remove a library item from a local library when you no longer need it.

To remove a library item from a library, you can call the `delete` method on the `Item` object and pass the library item ID as an argument. The item content is asynchronously removed from the storage.

You cannot remove items from a subscribed library. If you remove an item from a local library, the item is removed from the subscribed library when you perform a synchronization task on the subscribed library item.

Content Library Support for OVF Packages 7

Open Virtualization Format (OVF) is an industry standard that describes metadata about a virtual machine image in an XML format. An OVF package includes an XML descriptor file and optionally disk images, resource files (such as ISO files), manifest files, and certificate files.

With the vCloud Suite API, you can use the virtual machine (VM) and vApp templates from an OVF package in a content library to deploy VMs and virtual appliances on hosts, resource pools, and clusters. You can also use the API to create OVF packages in content libraries from virtual appliances and VMs on hosts, resource pools, and clusters.

When you create library items to store OVF packages, you must set the item type to `ovf`. You can use the objects and methods provided by the Content Library API to manage OVF packages. To comply with the specific standards of the OVF packages, the vCloud Suite API provides the `LibraryItem` class.

This chapter includes the following topics:

Using the Content Library Service to Handle OVF Packages

You can upload an OVF package to a library item by using the `UpdateSession` interface. The location of the OVF package determines whether you can pull the content from a URL or push the content directly to a content library.

For information about uploading content to library items, see "Upload a File from a Local System to a Library Item," on page 48 and "Upload a File from a URL to a Library Item," on page 50.

To download the files that are included in an OVF package to your local file system, use the `DownloadSession` interface. For more information, see "Download Files to a Local System from a Library Item," on page 51.

Upload an OVF Package from a URL to a Library Item

You can upload an OVF package from a Web server to a library item.

Procedure

1 Create an empty library item.

2 Create an update session object.

3 Create an `AddSpec` object to describe the properties and the upload location of the descriptor file of the OVF package.

4 Link the AddSpec object to the update session.

   All files that are included in the OVF package are automatically uploaded.

5 Complete the asynchronous transfer.

Java Example of Uploading an OVF Package from a URL to a Library Item

This example shows how you can upload an OVF package from a URL to a library item.

```
import com.vmware.content.library.Item;
import com.vmware.content.library.ItemModel;
import com.vmware.content.library.item.UpdateSessionModel;
import com.vmware.content.library.item.UpdateSession;
import com.vmware.content.library.item.TransferEndpoint;
import com.vmware.content.library.item.updatesession.File;
import com.vmware.content.library.item.updatesession.FileTypes.AddSpec;
import com.vmware.content.library.item.updatesession.FileTypes.SourceType;
import java.util.UUID;

  // Create an empty library item to describe the virtual machine.
  ItemModel itemModel = new ItemModel();
  itemModel.setName("ubuntu-vm");
  itemModel.setDescription("ubuntu 7.0");
  itemModel.setLibraryId(myLibraryId);
  itemModel.setType("ovf");
  String clientToken = UUID.randomUUID().toString();
  Item itemStub = myStubFactory.createStub(Item.class, myStubConfiguration);
  String itemId = itemStub.create(clientToken, itemModel);

  // Create an update session.
  UpdateSessionModel updateSessionModel = UpdateSessionModel();
  updateSessionModel.setLibraryItemId(itemId);
  clientToken = UUID.randomUUID().toString();
  UpdateSession updateSessionStub = myStubFactory.createStub(UpdateSession.class,
myStubConfiguration);
  String sessionId = updateSessionStub.create(clientToken, updateSessionModel);

  // Create a file specification for the OVF envelope file.
  AddSpec fileSpec = new AddSpec();
  fileSpec.setName("ubuntu.ovf");
  fileSpec.setSourceType(SourceType.PULL);
  TransferEndpoint endpoint = new TransferEndpoint();
  endpoint.setUri("http://www.example.com/images/ubuntu.ovf");
  fileSpec.setSourceEndpoint(endpoint);

  // Link the file specification to the update session.
  File updateFileStub = myStubFactory.createStub(File.class, myStubConfiguration);
  updateFileStub.add(sessionId, fileSpec);

  // Initiate the asynchronous transfer.
  updateSessionStub.complete(sessionId);
```

Upload an OVF Package from a Local File System to a Library Item

You can upload an OVF package from a local file system. This procedure describes how to use the `AddSpec` object after you have created a library item and initiated an update session.

Procedure

1 Create a library item.

2 Create an update session.

3 Create an `AddSpec` object to describe the properties and the upload location of the descriptor file of the OVF package.

4 Link the `AddSpec` object to the update session.

5 Create an `AddSpec` object for each VMDK file included in the OVF package.

6 Add all `AddSpec` objects to the update session.

Steps 5 and 6 must be repeated for each VMDK file included in the OVF package.

7 Initiate the upload operation.

8 Complete the update session.

9 Delete the session.

Java Example of Uploading an OVF Package to a Library Item from Your Local File System

This example shows how to upload an OVF package to a library item from your local file system.

```
import com.vmware.content.library.Item;
import com.vmware.content.library.ItemModel;
import com.vmware.content.library.item.UpdateSessionModel;
import com.vmware.content.library.item.UpdateSession;
import com.vmware.content.library.item.TransferEndpoint;
import com.vmware.content.library.item.updatesession.File;
import com.vmware.content.library.updatesession.FileTypes.AddSpec;
import com.vmware.content.library.updatesession.FileTypes.SourceType;
import java.util.UUID;

  // Create an empty library item to describe the virtual machine.
  ItemModel itemModel = new ItemModel();
  itemModel.setName(“ubuntu-vm”);
  itemModel.setDescription(“ubuntu 7.0”);
  itemModel.setLibraryId(myLibraryId);
  itemModel.setType(“ovf”);
  String clientToken = UUID.randomUUID().toString();
  Item itemStub = myStubFactory.createStub(Item.class, myStubConfiguration);
  String itemId = itemStub.create(clientToken, itemModel);

  // Create an update session.
  UpdateSessionModel updateSessionModel = UpdateSessionModel();
  updateSessionModel.setLibraryItemId(itemId);
  clientToken = UUID.randomUUID().toString();
  UpdateSession updateSessionStub = myStubFactory.createStub(UpdateSession.class,
myStubConfiguration);
  String sessionId = updateSessionStub.create(clientToken, updateSessionModel);
```

```
// Create a file spec for the OVF envelope file.
AddSpec fileSpec = new AddSpec();
fileSpec.setName("ubuntu.ovf");
fileSpec.setSourceType(SourceType.PULL);
TransferEndpoint endpoint = new TransferEndpoint();
endpoint.setUri("http://www.example.com/images/ubuntu.ovf");
fileSpec.setSourceEndpoint(endpoint);

// Link the file spec to the update session.
File updateFileStub = myStubFactory.createStub(File.class, myStubConfiguration);
updateFileStub.add(sessionId, fileSpec);

// Initiate the asynchronous transfer.
updateSessionStub.complete(sessionId);
```

Using the LibraryItem Service to Execute OVF-Specific Tasks

You can deploy virtual machines and vApps on hosts, clusters, and resource pools in your environment. You use the VM templates and vApp templates from an OVF package that is stored as a content library item.

With the vCloud Suite API, you can use the `LibraryItem` service to deploy virtual machines and virtual applications from library items that contain OVF packages. You can also use the `LibraryItem` vCloud Suite service to create library items from existing virtual machines and virtual appliances.

Deploy a Virtual Machine or Virtual Appliance from an OVF Package in a Content Library

You can use the `LibraryItem` service to deploy a virtual machine or virtual appliance on a host, cluster, or resource pool from a library item.

Procedure

1. Create a `DeploymentTarget` instance to specify the deployment location of the virtual machine or virtual appliance.
2. Instantiate the `ResourcePoolDeploymentSpec` class to define all necessary parameters for the deployment operation.

   For example, you can assign a name for the deployed virtual machine or virtual appliance, and accept the End User License Agreements (EULAs) to complete the deployment successfully.
3. (Optional) Retrieve information from the descriptor file of the OVF package and use the information during the OVF package deployment.
4. Invoke the `deploy` method on the `LibraryItem` service.
5. Verify the outcome of the deployment operation.

Java Example of Deploying a Virtual Machine from a Library Item in a Resource Pool

This example shows how to deploy a virtual machine from a local library item in a resource pool. You can also see how to verify the results of the deployment operation.

```
import com.vmware.content.library.Item;
import com.vmware.vcenter.ovf.LibraryItemTypes.DeploymentResult;
import com.vmware.vcenter.ovf.LibraryItemTypes.DeploymentTarget;
import com.vmware.vcenter.ovf.LibraryItemTypes.OvfSummary;
```

```
import com.vmware.vcenter.ovf.LibraryItemTypes.ResourcePoolDeploymentSpec;
import com.vmware.vcloud.suite.samples.common.ServiceManager;
import com.vmware.vcloud.suite.samples.vim.helpers.VimUtil;
import com.vmware.vim25.ManagedObjectReference;
import java.util.UUID;

...

// Create a virtual machine deployment specification to accept any network resource.
ResourcePoolDeploymentSpec deploymentSpec = new ResourcePoolDeploymentSpec();
String vmName = "MyVirtualMachine";
deploymentSpec.setName(vmName);
deploymentSpec.setAcceptAllEULA(true);


// Create a deployment target specification to accept any resource pool.
ServiceManager serviceManager = getServiceManager(null);
String clusterName = "myCluster";
ManagedObjectReference clusterMoRef = VimUtil.getCluster(serviceManager.getVimPortType(),
serviceManager.getServiceContent(), clusterName);
DeploymentTarget deploymentTarget = new DeploymentTarget();
deploymentTarget.setResourcePoolId(clusterMoRef.getValue());


// Retrieve the library items OVF information and use it for populating the
// deployment spec instance.
LibraryItem libItemStub = stubFactory.createStub(LibraryItem.class, myStubConfiguration);
OvfSummary ovfSummary = libItemStub.filter(libItemId, deploymentTarget);
deploymentSpec.setAnnotation(ovfSummary.getAnnotation());
String clientToken = UUID.randomUUID().toString();
DeploymentResult result = libItemStub.deploy(clientToken,libItemId,
                                                        deploymentTarget,
                                                        deploymentSpec);

// Verify the status of the resource deployment.
System.out.printf("Resource Type=%s (ID=%s) status: ",
                  result.getResourceId().getType(),
                  result.getResourceId().getId());

  if (result.getSucceeded() == true) {
      System.out.println("Resource instantiated.");

  } else {
      System.out.println("Instantiation failed.");
  }
```

Create an OVF Package in a Content Library from a Virtual Machine

You can create library items from existing virtual machines or virtual appliances. Use those library items later to deploy virtual machines and virtual appliances on hosts and clusters in the vCloud Suite environment.

Procedure

1 Create a `DeployableIdentity` instance to specify the source virtual machine or virtual appliance to be captured in an OVF package.

2 Create a `CreateTarget` instance to identify the content library where the OVF package is stored.

3 Create a `CreateSpec` instance to specify the properties of the OVF package.

4 Initiate a synchronous create operation by invoking the `create` function of the `LibraryItem` service.

5 Verify the results of the `create` operation.

Java Example of Creating an OVF Package in a Content Library from a Virtual Machine

This example shows how to capture a virtual machine in an OVF package and store the file in a new library item in a specified library.

```
import com.vmware.vcenter.ovf.LibraryItem;
import com.vmware.vcenter.ovf.LibraryItemTypes;
import com.vmware.vcenter.ovf.LibraryItemTypes.CreateTarget;
import com.vmware.vcenter.ovf.LibraryItemTypes.DeployableIdentity;
import java.util.UUID;

// Specify the resource to be captured.
LibraryItemTypes.DeployableIdentity deployableIdentity = new
LibraryItemTypes.DeployableIdentity();
deployableIdentity.setType("VirtualMachine");
deployableIdentity.setId("vm-32");

// Create a target spec to identify a library to hold the new item.
LibraryItemTypes.CreateTarget createTarget = new LibraryItemTypes.CreateTarget();
createTarget.setLibraryId(myLibraryId);

// Specify OVF properties.
LibraryItemTypes.CreateSpec createSpec = new LibraryItemTypes.CreateSpec();
createSpec.setName("snap-32");
createSpec.setDescription("Snapshot of VM-32");

// Initiate synchronous capture operation.
LibraryItem itemStub = myStubFactory.createStub(LibraryItem.class, myStubConfiguration);
String clientToken = UUID.randomUUID().toString();
LibraryItemTypes.CreateResult result = itemStub.create(clientToken, deployableIdentity,
createTarget, createSpec);

// Verify capture status.
System.out.printf("Resource Type=%s (ID=%s) status:",
                  deployableIdentity.getType(),
                  deployableIdentity.getId());
  if (result.getSucceeded() == true) {
     System.out.println("Resource captured.");

  }else {
     System.out.println("Capture failed.");
  }

  if (result.getError() != null) {

    for (OvfError error : result.getError().getErrors()) {
        System.out.printf("Error: %s", error.getMessage().toString());
    }
```

```
  for (OvfWarning warning : result.getError().getWarnings()) {
      System.out.printf("Warning: %s", warning.getMessage().toString());
  }

  for (OvfInfo info : result.getError().getInformation()) {
      List<LocalizableMessage> messages = info.getMessage();
  }

  for (LocalizableMessage message : messages) {
    System.out.printf("Message: %s", message.toString());
  }
}
```

Tagging Service

8

The vCloud Suite Tagging Service supports the definition of tags that you can associate with vSphere objects or vCloud Suite resources. vSphere has a tagging feature but no public API to manage tags. With the vCloud Suite SDK, you can manage tags programmatically.

For example, to tag your VMs by guest operating system type, you might create a category called **`operating system`**, and specify that it applies to VMs only. You might also specify that only a single tag can be applied to a VM at any time. The tags in this category might be **`Windows`**, **`Linux`**, and **`Mac OS`**.

Creating Tags

When you create a tag, you create a tag category and create a tag under the category. After you create the tag, you can associate the tag with an object.

Tags and categories have global scope. The Platform Services Controller stores tags and categories makes them available to any vCenter Server system that is registered with the Platform Services Controller.

Creating a Tag Category

You create tags in the context of a tag category. You must create a category before you can add tags within that category.

A tag category has the following properties:

- name
- description
- cardinality, or how many tags it can contain
- the types of elements to which the tags can be assigned

You can associate tags with both vSphere API managed objects and VMware vCloud Suite API resources.

Java Example of Creating a Tag Category

This example is based on code in the `TaggingWorkflow.java` sample file. This file is located in the following vCloud Suite SDK for Java directory: `client/samples/java/com/vmware/vcloud/suite/sample/interop`

The category `create()` function returns an identifier that you use when you create a tag for that category. The empty set for the associable types indicates that any object type can be associated with a tag in this category.

```
import com.vmware.cis.tagging.Category;
import com.vmware.cis.tagging.CategoryMadel;
import com.vmware.cis.tagging.CategoryTypes;
import com.vmware.cis.tagging.Tag;
import com.vmware.cis.tagging.TagCategory;
import com.vmware.cis.tagging.TagAssociation;

Category categoryStub = myStubFactory.createStub(Category.class,
                                                  myStubConfiguration);

// Set up a tag category create spec.
CategoryTypes.CreateSpec createSpec = new CategoryTypes.CreateSpec();
createSpec.setName("favorites");
createSpec.setDescription("My favorite virtual machines.");
createSpec.setCardinality(CategoryModel.Cardinality.MULTIPLE);
Set<String> associableTypes = new HashSet<String>();
createSpec.setAssociableTypes(associableTypes);

String newCategoryId = categoryStub.create(createSpec);
```

Creating a Tag

After you create a tag category, you can create tags within that category

A tag has the following properties:

- name
- description
- category ID

Java Example of Creating a Tag

This example creates a tag specification and then uses it to create the tag. The tag specification references the category identifier that was returned from the category create operation. Use the returned tag identifier for subsequent operations on the tag.

This example is based on code in the `TaggingWorkflow.java` sample file. This file is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/com/vmware/vcloud/suite/sample/interop/filepath>`

```
import com.vmware.cis.tagging.Tag;
import com.vmware.cis.tagging.TagMadel;
import com.vmware.cis.tagging.TagTypes;
import com.vmware.cis.tagging.Tag;
//import com.vmware.cis.tagging.TagCategory;
//import com.vmware.cis.tagging.TagAssociation;

Tag tagStub = myStubFactory.createStub(Tag.class,
                                       myStubConfiguration);

// Set up a tag create spec.
TagTypes.CreateSpec spec = new TagTypes.CreateSpec();
spec.setName("red");
spec.setDescription("My favorite color");
spec.setCategoryId(newCategoryId);

String tagId = tagStub.create(spec);
```

Creating Tag Associations

After you create a tag category and create a tag within the category, you can associate the tag with a vSphere managed object or a vCloud Suite resource. An association is a simple link that contains no data of its own. You can enumerate objects that are attached to a tag or tags that are attached to an object.

Tag associations are local to a vCenter Server instance. When you request a list of tag associations from a vCenter Server system, it enumerates only the associations that it has stored.

When you associate a tag with an object, the object's type must match one of the associable types specified for the category to which the tag belongs.

- Assign the Tag to a Content Library on page 65

  After you create a tag, you can assign the tag to a vCloud Suite resource.
- Assign a Tag to a Cluster on page 66

  After you create a tag, you can assign the tag to a vSphere managed object. Tags make the inventory objects in your virtual environment more sortable and searchable.

Assign the Tag to a Content Library

After you create a tag, you can assign the tag to a vCloud Suite resource.

Procedure

1 Construct a dynamic object identifier for the library.

  The dynamic identifier includes the type and ID of the object.

2 Attach the tag to the content library.

Java Example of Assigning a Tag to a Content Library

This example assigns a tag to a content library.

```
import com.vmware.vapi.std.DynamicID;
import com.vmware.cis.TagAssociation;

// 1 – Create a dynamic type ID for the content library.
DynamicID libraryDynamicId = new DynamicID(Library.RESOURCE_TYPE,
                                           myLibrary.getId());

// 2– Attach the tag to the ClusterComputeResource managed object.
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(myLibrary.getId(),
                          libraryDynamicId);
```

Assign a Tag to a Cluster

After you create a tag, you can assign the tag to a vSphere managed object. Tags make the inventory objects in your virtual environment more sortable and searchable.

This procedure describes the steps for applying tag a to a cluster object in your inventory.

Prerequisites

Obtain the managed object identifier for the specified cluster.

To get the managed object identifier of the `ClusterComputeResource`, you must access vCenter Server by using the vSphere Web Services API. For more information about how to access Web Services, see "Create a Web Services Session," on page 29.

Procedure

1 Construct a dynamic object identifier for the cluster.

   The dynamic identifier includes the type and ID of the managed object reference.

2 Attach the tag to the cluster.

Java Example: Assigning the Tag to a `ClusterComputeResource`

Most of this example is based on code in the `TaggingWorkflow.java` sample file, which is located in the following vCloud Suite SDK for Java directory:
`client/samples/java/com/vmware/vcloud/suite/samples/interop/`

```
import com.vmware.vcloud.suite.samples.vim.helpers.VimUtil;
import com.vmware.vapi.std.DynamicID;
import com.vmware.cis.TagAssociation;

// 1 – Determine the MOID of the ClusterComputeResource from its name.
ManagedObjectReference clusterMoRef = VimUtil.getCluster(vimPort,
                                                         serviceContent,
                                                         myClusterName);

// 2 – Create a dynamic type ID for the cluster.
DynamicID clusterDynamicId = DynamicID(clusterMoRef.getType(),
                                       clusterMoRef.getValue());

// 3 – Attach the tag to the ClusterComputeResource managed object.
```

```
TagAssociation tagAssociationStub = myStubFactory.createStub(TagAssociation.class,
                                                             myStubConfig);
tagAssociationStub.attach(tagId,
                          clusterDynamicId);
```

Updating a Tag

To update a tag, you must create an update spec for the tag. In the update spec, you set values for the fields to be changed, and omit values for the other fields. When you do an update operation using the update spec, only the fields that contain values are changed.

For example, you might use a timestamp in a tag description to identify a resource's last reconfiguration. After reconfiguring the resource, you update the tag description to contain the current time.

Java Example of Updating a Tag Description

This example adds timestamp in a tag description to identify when a resource was last reconfigured. The tag description is updated with the timestamp after the resources is reconfigured.

```
import java.util.Date;
import java.text.Dateformat;

String newDateTime = Dateformat.getDateInstance().format(new Date());
String newDescription = String.format("Server tag updated at (%s).", newDateTime);

TagTypes.UpdateSpec updateSpec = new TagTypes.UpdateSpec();
updateSpec.setDescription(newDescription);
tagStub.update(myTagId, updateSpec);
```

Index