VMware Storage Policy SDK Programming Guide

11 OCT 2022

VMware vSphere 8.0
vCenter Server 8.0
VMware ESXi 8.0

vmware
by Broadcom

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

https://docs.vmware.com/

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

Contents

About This Book

The *VMware Storage Policy Programming Guide* describes how to use the VMware® Storage Policy API.

The VMware Storage Policy SDK supports the development of vCenter clients that use storage profiles for virtual machine configuration. This method of storage administration is called storage policy based management (SPBM).

Revision History

This book is revised with each release of the product or when necessary. The following table summarizes significant changes in each version of this book.

Revision Date	Description
11 OCT 2022	vSphere 8.0 release. Minor cleanup.
02 APR 2020	Improved illustrations for vSphere 7.0.
17 Apr 2018	Revised for vSphere 6.7 with new chapter about VVol storage profiles.
16 Nov 2016	Revised for vSphere 6.5 with minor updates.
12 Mar 2015	Revised for vSphere 6.0 with new chapters about Tag, VM, and vSAN storage profiles.
12 Sep 2013	First version of this manual for vSphere 5.5 with information about storage profiles.

Intended Audience

This book is intended for anyone who needs to develop applications using the VMware Storage Policy SDK. An understanding of Web Services technology and some programming background in Java is required.

Sample Code

The VMware Storage Policy SDK includes Java programs to list storage profiles, view a specific profile, create a new storage profile, associate a storage profile with a VM (when creating, cloning, or relocating the VM), check compliance of the VM with its storage profile, and delete a storage profile.

API Reference

Many of the SPBM interfaces start with the letters Pbm, for policy based management. To find an API reference information, go to https://code.vmware.com/sdksand click **vSphere Management SDK** then click **VMware Storage Policy API Reference**. There are Pbm interfaces in all categories: managed objects, data objects, enumerated types, fault types, methods, and so forth.

VMware Storage Policies

1

Storage Policy Based Management (SPBM) helps administrators automate VM provisioning on appropriate storage devices with requested data services. This avoids laborious manual provisioning of storage for individual VMs. For object-based datastores like vSAN and VVol, vendors formulate metadata describing storage traits, and advertise storage capabilities for profile queries.

Read the following topics next:

Storage Capabilities

The storage provider describes capabilities of a storage array, which the storage policy service obtains and presents to the administrator to assist with VM provisioning.

On object-oriented datastores such as vSAN and VVol datastores, storage capabilities originate from software written by the vendor, called a VASA provider. VASA is an abbreviation of vSphere API for storage awareness. The storage policy service can match storage capabilities with VM storage policies formulated in vSphere.

For VVol datastores: storage capabilities may include array type, vendor, RAID level, read latency, write latency, snapshot information, backup frequency, replication, caching, compression, deduplication, and high availability.

For vSAN datastores: storage capabilities may include RAID type, failures to tolerate, disk stripes per object, flash read cache, IOPS limit, encryption, compression, and deduplication.

For VMFS and NFS datastores: administrators can select the VMware encryption storage policy. VM encryption is implemented on the ESXi host rather than on the storage array.

For tag-based policies: administrators can use the vSphere Web Client to define storage policy tags.

Virtual Machine Storage

Virtual machine configuration and virtual disk data reside in datastores.

Virtual machine configuration is stored in a file with the `.vmx` file extension. VM home files also include other files for virtual machine operation, such as a log file (`.log`), virtual firmware (`.nvram`), paging file (`.vmem`), and snapshot data files (`.vmsd`).

Virtual machine data is stored in virtual disks, in files with the `.vmdk` file extension.

Storage policies allow you to distinguish between virtual machine configuration and data files and to specify storage locations based on the needs of each.

Storage Capability Profiles

To use a storage capability profile, you define storage requirements and associate a virtual machine with a storage policy. When you create a VM, vCenter Server correlates its VM storage policy with the storage policy service to determine an appropriate location for VM home and virtual disk files.

A storage policy consists of a set of subprofiles. Each subprofile defines a set of storage capabilities, and corresponds to a numbered Rule Set in the vSphere Web Client. Several storage policies pre-exist in vSphere, including:

- The VM Encryption Policy is a sample storage policy for VM Home and virtual disk encryption.
- The vSAN Default Storage Policy can be easily modified, but is the storage policy used by default for vSAN datastores.
- The VVol No Requirements Policy allows the storage array, through the VASA provider, to determine the best placement strategy for VM Home and virtual disks.

In the vSphere Web Client, all of these storage policies, and ones you create, have a **Check Compliance** button to verify that virtual machines assigned to that storage policy are correctly placed on storage devices.

Storage Policy Operations

The Storage Policy API is implemented with Web Services Description Language (WSDL), as an XML file describing how to communicate with it. The WSDL is usable with toolkit bindings by many programming languages, including Java.

SPBM Managed Objects

The Storage Policy API is structured like the vSphere API, and is likewise suitable for Java programming. Managed object types include the following:

- `PbmProfileProfileManager` supports operations on virtual machine storage profiles.
- `PbmPlacementSolver` identifies locations that support requested capabilities for storing virtual machine files.
- `PbmComplianceManager` verifies compliance of virtual machine and virtual disk requirement profiles.
- `PbmReplicationManager` deals with replication of virtual machine and virtual disk requirement profiles.
- `PbmServiceInstance` is the root object of Storage Policy service, created connecting to SPBM.

Storage Policy API and vSphere API

The following table shows the correspondence between SPBM managed objects and vSphere API methods.

Table 1-1. Storage Policy Operations and Virtual Machine Provisioning

SPBM Operation (Storage Policy API)	Virtual Machine Provisioning (vSphere API)
Use the `PbmProfileProfileManager` methods to create and update storage profiles.	Associate storage profiles with virtual machines and virtual disks. See the description of the vSphere API data object properties `VirtualMachineConfigSpec.vmProfile` and `FileBackedVirtualDiskSpec.profile` in the vSphere API Reference. You can also use the vSphere Web Client to associate a storage profile with a virtual machine or virtual disk.
Use the `PbmPlacementSolver` methods to identify candidate datastores for storage locations.	Specify the datastores when you create virtual machines and virtual disks. See the description of the vSphere API data object properties `VirtualMachineFileInfo.vmPathName` and `VirtualDeviceFileBackingInfo.datastore` in the vSphere API Reference.
Use the `PbmComplianceManager` methods to check compliance between storage requirements and capabilities.	After you associate a storage profile with a virtual machine or virtual disk, the Server will identify non-compliance if the datastore does not satisfy the requirements of the profile.

Access to the Storage Policy Server

You can access the VMware Storage Policy Server by connecting to a vCenter Server system and obtaining a vCenter session cookie.

The Storage Policy client API is described in the pbmService.wsdl file that is included in the VMware Storage Policy SDK. The API defines a set of request operations that you use to manipulate storage profiles. The VMware Storage Policy SDK includes Java bindings for the SPBM service WSDL.

To gain access to the Storage Policy Server, your client connects to a vCenter Server system and obtains the vCenter session cookie. Then you can use the vCenter session cookie to establish the connection with the Storage Policy Server. See Establish a Connection with the VMware Storage Policy Server.

After you establish a Storage Policy Server connection, your client uses language-specific Web Services access objects and the `PbmServiceInstance` and `PbmServiceInstanceContent` objects to access the Storage Policy managed objects and their methods.

The Storage Policy Web Services access objects are language-specific API binding objects that are generated from the Storage Policy WSDL. The VMware Storage Policy SDK contains JAXWS bindings to the Storage Policy API. The JAXWS bindings include the `PbmService` and `PbmPortType` Web Services access objects.

- `PbmService` – Provides access to the `PbmPortType` object and support for the Storage Policy Service connection.
- `PbmPortType` – Provides access to Storage Policy methods.

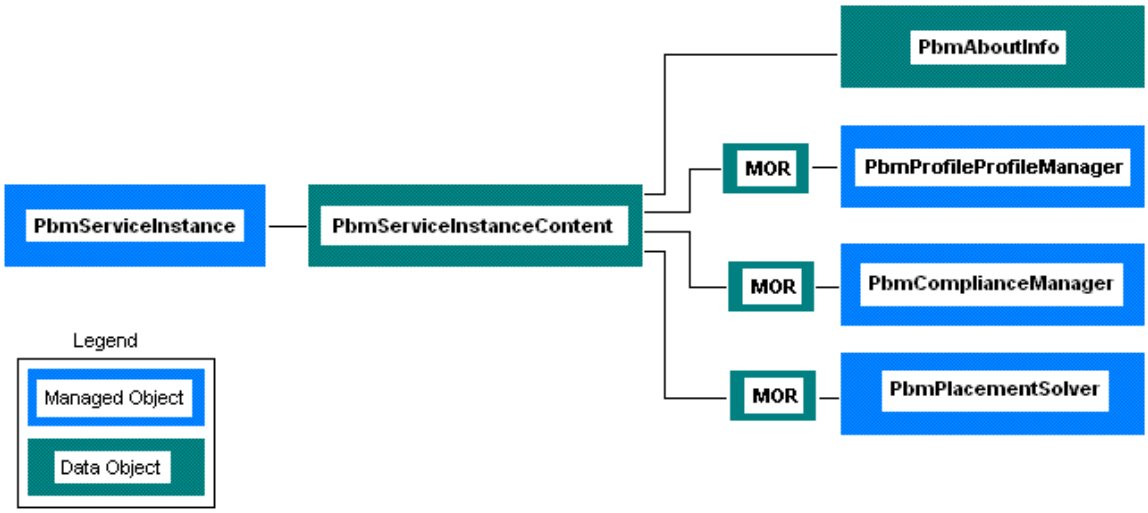
The following code fragment shows the sequence of calls that you use to obtain access to the Storage Policy API methods.

Example: Access to Storage Policy API Methods

```
import com.vmware.pbm.PbmService;
import com.vmware.pbm.PbmPortType;
import com.vmware.pbm.PbmServiceInstanceContent;
[...]
PbmService = new PbmService()
PbmPortType pbmPort = PbmService.getPbmPort()
PbmServiceInstanceContent pbmServiceContent = pbmPort.pbmRetrieveServiceContent
```

The following figure shows the resulting `PbmServiceInstanceContent` data object and the Storage Policy managed objects that provide access to Storage Policy services.

Figure 1-1. Storage Policy Service Instance Content



The `PbmServiceInstanceContent` object contains managed object references to the Storage Policy services. The set of Storage Policy services include the profile manager, placement solver, and compliance manager.

Service	Managed Object	Description
Profile Manager	`PbmProfileProfileManager`	Create and update VMware storage profiles to define storage requirements.
Placement Solver	`PbmPlacementSolver`	Identify candidate datastores for storage locations.
Compliance Manager	`PbmComplianceManager`	Check compliance between storage requirements and capabilities.

Storage Profile Queries

The Storage Policy API includes several methods that you can use to query for profiles and vSphere entities, such as datastores, virtual machines, and virtual disks.

The following table provides an overview of these methods. For more information, see the *Storage Policy API Reference*.

Table 1-2. Storage Profile API Query Methods

Method	Description
`PbmQueryAssociatedEntities`	Returns the virtual machine and disks that are associated with the given storage policies. With empty parameter, returns all virtual machine and disks that are associated with a storage policy.
`PbmQueryAssociatedEntity`	Returns references to entities associated with the specified profile.

Table 1-2. Storage Profile API Query Methods (continued)

Method	Description
`PbmQueryAssociatedProfile`	Returns profiles associated with the specified entity. The type of profile is determined by the type of entity that you specify. ■ If you specify a datastore, the method returns one or more capability (resource) profiles. ■ If you specify a virtual machine or virtual disk, the method returns one or more requirement profiles.
`PbmQueryAssociatedProfiles`	Returns `PbmQueryProfileResult` objects. Each result objet identifies an entity and one or more profiles. Profile type is determined by entity type. ■ If the entity is a datastore, the result object contains one or more capability (resource) profiles. ■ If the entity is a virtual machine or virtual disk, the result object contains one or more requirement profiles.
`PbmQueryByRollupComplianceStatus`	Returns all virtual machines for the given rollup compliance status.
`PbmQueryDefaultRequirementProfile`	Returns the default requirement profile ID for the given datastore. For legacy hubs, returns `null`.
`PbmQueryDefaultRequirementProfiles`	Returns the default profiles for the given datastores. For legacy datastores, the `defaultProfile` is set to null.
`PbmQueryMatchingHub` (deprecated)	Finds matching placement hubs for the specified requirements profile. Returns only hubs that match the profile. If this method is called for `VVolDefaultProfile`, then all VVol containers are returned as matching.
`PbmQueryMatchingHubWithSpec` (deprecated)	Finds matching placement hubs based on a profile creation specification. This method returns only those hubs that match the specification.
`PbmQueryProfile`	Returns requirement profiles or resource profiles, or both.
`PbmQueryReplicationGroups`	Returns identifiers for replication groups associated with virtual machines, virtual disks, or virtual machines and all their disks. If the query is performed for a virtual machine and all its disks are `virtualMachineAndDisks`, an entry per disk and one for the virtual machine configuration are returned.
`PbmQuerySpaceStatsForStorageContainer`	Retrieves space statistics of a datastore.

VMware Storage Policy SDK

The VMware Storage Policy SDK is distributed as part of the VMware vSphere Management SDK.

When you extract the contents of the distribution, the VMware Storage Policy SDK is located in the `spbm` sub-directory.

```
VMware-vSphere-SDK-build-num
     eam
     sms-sdk
     spbm
```

```
        docs
            java
                JAXWS/samples/javadoc/index.html
            ReferenceGuide
                index.html
        java
            JAXWS
                lib
                samples
                build and run scripts
        wsdl
            pbmService.wsdl
            pbm.wsdl
    ssoclient
    vsphere-ws
```

The following table shows the locations of the contents of the VMware Storage Policy SDK.

Table 1-3. VMware Storage Policy SDK Contents

VMware Storage Policy SDK Component	Location
JAX-WS VMware Storage Policy client binding	`spbm/java/JAXWS/lib`
Java Storage Policy samples	`spbm/java/JAXWS/samples/com/vmware/spbm/samples/`
Java Storage Policy Server connection sample	`spbm/java/JAXWS/samples/com/vmware/spbm/connection/`
VMware Storage Policy API Reference	`spbm/docs/ReferenceGuide/index.html`
Documentation for example code	`spbm/docs/java/JAXWS/samples/javadoc/index.html`
WSDL files	`spbm/wsdl`

VMware Storage Policy SDK Examples

The VMware Storage Policy SDK contains Java examples that show how to create and use VMware storage policies.

This manual describes examples from the VMware Storage Policy SDK. It also describes examples from the vCenter Single Sign-On SDK that support the client connection to the Storage Policy Server. This manual includes the following single sign-on examples:

- Chapter 9 vCenter Single Sign-On Client Example. This example shows how to obtain a holder-of-key token from the ESXivCenter Single Sign-On Server.
- Chapter 10 vCenter LoginByToken Example. This example shows how to use the token to login to vCenter Server.

The following table lists the sample files in the VMware Storage Policy SDK.

Table 1-4. VMware Storage Profile SDK Sample File

Location	Examples	Description
SDK/spbm/java/JAXWS/samples/com/vmware/spbm/samples/		
	AboutInfo.java	Obtains identifying data about the Storage Policy Server.
	CheckCompliance.java	Checks the compliance of profiles associated with virtual machines and virtual disks.
	CreateProfile.java	Creates a requirement profile.
	CreateVSANProfile.java	Creates a new storage profile with one rule-set based on vSAN capabilities.
	DeleteProfile.java	Deletes a requirement profile.
	EditProfile.java	Adds or deletes subprofiles from a tag-based storage profile.
	FcdAssociateProfile.java	Attaches a first class disk (FCD) to a virtual machine and associates the given storage profile with the FCD.
	ListProfiles.java	Retrieves all of the storage profiles known to the system.
	ViewProfile.java	Prints the contents of a tag-based storage profile.
	VMClone.java	Deploys multiple instances of a virtual machine template to a datacenter. The clone specification has an associated storage profile.
	VMCreate.java	Creates a virtual machine. The virtual machine configuration specification has an associated storage profile.
	VMRelocate.java	Used to relocate a virtual machine's virtual disks to a datastore compliant with the given storage profile.
SDK/spbm/java/JAXWS/samples/com/vmware/spbm/connection/		
	BasicConnection.java	Establishes an authenticated session with a VMware SSO Server, vCenter Server, and Storage Policy Server.
	ConnectedServiceBase.java	Connection base class for client application implementations.
	Connection.java	Storage Policy sample support; utility class that sets up a Storage Policy Server connection.
	ConnectionException.java	Base exception class for exceptions thrown by connection classes.
	ConnectionMalformedUrlException.java	URL exception.

Table 1-4. VMware Storage Profile SDK Sample File (continued)

Location	Examples	Description
	KeepAlive.java	Keep-alive utility class; maintains the vCenter Server connection.
	VcSessionHandler.java	Utility class; inserts vCenter session cookie into SOAP header.

Storage Policy Server Connection 2

The connection between a Storage Policy client and the Storage Policy Server is based on the client's connection with a vCenter Server system.

Read the following topics next:

- About Storage Policy Server Sessions
- Establish a Connection with the VMware Storage Policy Server
- Create the Storage Policy Server Connection
- Establish the vCenter Session Connection for the Local Instance

About Storage Policy Server Sessions

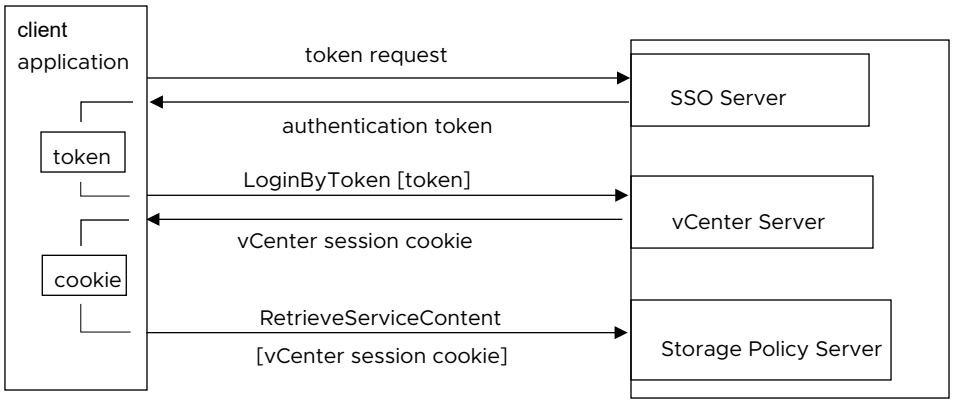
A vCenter Server client uses an HTTP session cookie to maintain a persistent connection with the Server. A Storage Policy client uses the vCenter Server session cookie to establish the connection with the Storage Policy Server.

A client performs the following operations to establish vCenter Server and Storage Policy Server sessions.

- Obtain a SAML token from the VMware SSO Server.
  See Chapter 9 vCenter Single Sign-On Client Example.
- Use the SAML token to login to the vCenter Server.
  See Chapter 10 vCenter LoginByToken Example.
- Use the `RetrieveServiceContent` method to send the session cookie to the Storage Policy Server and establish the connection with the Server.

The following figure shows a representation of the server connections and operations involved in establishing a Storage Policy Server connection.

Figure 2-1. Storage Policy Server Connection



Establish a Connection with the VMware Storage Policy Server

Use the session cookie from the vCenter Server session to establish the Storage Policy session. The session cookie represents the authenticated vCenter Server session, which is based on the SSO token.

The following code fragments establish connections both with the vCenter Server and the Storage Policy Server. These examples are based on the `BasicConnection` sample, which is located in the Storage Policy SDK connection sample directory.

`SDK/spbm/java/JAXWS/samples/com/vmware/spbm/connection/BasicConnection.java`

The `BasicConnection` sample uses an instance of the `LoginByTokenSample` class. See Chapter 10 vCenter LoginByToken Example. The `LoginByToken` example saves the HTTP cookie produced during the intial connection sequence and then restores the cookie after the vCenter Server connection has been established. Although the `LoginByToken` example creates a vCenter Server connection, the `BasicConnection` sample establishes its own connection with the vCenter Server. A different implementation might integrate those capabilities to reduce the number of vCenter Server connections.

Server URLs for Basic Connection

The `BasicConnection` sample creates connections to three VMware Servers.

- SSO Server
- vCenter Server
- Storage Policy Server

In the example configuration, the SSO and Storage Policy Servers are located on the same system as the vCenter Server instance. In other configurations, the SSO Server might be located on a different server.

Table 2-1. VMware Server URLs

VMware Server	URL
vCenter Server	https://*server-name*|*IPaddress*/sdk/vimService
SSO Server	https://*server-name*|*IPaddress*/sts/STSService
Storage Policy Server	https://*server-name*|*IPaddress*/pbm

Create the Storage Policy Server Connection

The following code fragment uses a vCenter session cookie to create a Storage Policy Server session.

1 Extract the actual cookie value from the `name=value` expression in the cookie string obtained from the vCenter session connection.

2 Create a `PbmService` object.

3 Set up a header handler to support adding the vCenter session cookie to the Storage Policy Server connection.

4 Retrieve the `PbmPort` object for access to the Storage Policy API methods.

5 Retrieve the request context and set the endpoint to the Storage Policy Server URL.

6 Call the `PbmRetrieveServiceContent` method to establish the HTTP connection to the Storage Policy Server.

Example: Storage Policy Server Connection

```
// 1. Set the extracted cookie in the PbmPortType
//
// Need to extract only the cookie value
String[] tokens = cookieVal.split(";");
tokens = tokens[0].split("=");
String extractedCookie = tokens[1];

// 2. Create a PbmService object.
pbmService = new PbmService();

// 3. Setting the header resolver for adding the VC session cookie to the
// requests for authentication
HeaderHandlerResolver headerResolver = new HeaderHandlerResolver();
headerResolver.addHandler(new VcSessionHandler(extractedCookie));
pbmService.setHandlerResolver(headerResolver);

// 4. Retrieve the PbmPort object for access to the Storage Policy API
pbmPort = pbmService.getPbmPort();

// 5. Set the Storage Policy Server endpoint
Map<String, Object> pbmCtxt = ((BindingProvider) pbmPort).getRequestContext();
```

```
pbmCtxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);
pbmCtxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, spbmurl.toString());

// 6. Retrieve the service content (creates the connection)
pbmServiceContent = pbmPort.pbmRetrieveServiceContent(getPbmServiceInstanceReference());
```

Establish the vCenter Session Connection for the Local Instance

The following code fragment sets up the HTTP connection with the vCenter Server instance.

1 Retrieve the `VimPort` interface. This provides access to the vSphere API methods.

2 Retrieve the request context and set the vCenter Server endpoint address in the request context.

3 Set the session cookie in the request context. The cookie (`cookieVal`) is obtained from the Chapter 10 vCenter LoginByToken Example.

4 Call the `RetrieveServiceContent` method to establish the HTTP connection with the vCenter Server instance.

Example: vCenter Server Connection

```
// 1. Retrieve the VimPort interface.
vimService = new VimService();
vimPort = vimService.getVimPort();

// 2. Retrieve the request context and set the vCenter Server endpoint.
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcurl.toString());
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

// 3. Put the extracted vCenter session cookie into the VimPortType request header.
Map<String, List<String>> headers =
     (Map<String, List<String>>) ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
     headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookieVal));
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);

// 4. Retrieve the vCenter Server service content. (Establishes the HTTP connection)
vimServiceContent = vimPort.retrieveServiceContent(this.getVimServiceInstanceReference());
```

vVol Based Storage Profiles

3

Virtual Volumes (vVols) help with granularity in shared storage by providing a way to designate a specific storage policy for each virtual machine or virtual disk. Vendors formulate metadata describing their storage arrays, advertise capability profiles in a VASA provider, so SPBM can query storage capabilities on request. VASA is an abbreviation of vSphere API for storage awareness.

Read the following topics next:

- vVol Storage Policy Architecture
- SPBM Calling Sequence for vVols
- SPBM Data Objects

vVol Storage Policy Architecture

Storage Policy Based Management (SPBM) helps vSphere and storage administrators automate the provisioning of virtual machines on shared vVol storage with desired data services.

Storage Management Service (SMS) and SPBM run as part of the Storage Policy Service (SPS), which is a daemon running in vCenter Server to handle storage subsystems. SPS communicates with vendor-provided VASA providers, which implement about a dozen SPBM related functions to advertise storage capabilities and manage storage requests.

On the storage array, a virtual machine comprises a Config vVol with descriptors, a Swap vVol for memory, and a Data vVol for each virtual disk. Data to and from a storage container flows through a protocol endpoint, which can support either SAN or NAS protocols. Snapshots, space efficiency, replication, and other storage features may be handled natively by the storage array.

Typically storage administration involves a set of rule sets where platinum or gold policy offers the best quality of service, silver policy in the middle, and bronze policy the fewest features. A vSphere administrator can configure these policies using the vSphere Client connected to vCenter Server. In vSphere there is a predefined No Requirements policy for vVols.

SPBM Calling Sequence for vVols

When managing vVol storage, SPBM uses a specific calling sequence.

After a VASA provider registers with vSphere, SPBM makes the following calls:

1 `queryCapabilityMetadata` to get the capability metadata exposed by the VASA provider, for example RAID level, compression, encryption, caching, or replication.

2 `queryResourceMetadata` to get a list of available storage resources, expressed as capability metadata.

3 `queryMatchingContainer` to get a list of compatible storage containers for a given policy. The response from the VASA provider refers to resources returned by the `queryResourceMetadata` call. There will also be provisioning requests before `queryComplianceResult` is called - not by SPBM, but from the ESXi host.

4 `queryComplianceResult` to check compliance with storage policies.

5 For version 3.0 VASA providers, `queryPolicy` is called if required to retrieve the Replication Group ID of the storage device for a vVol. ESXi hosts call it to query the policy of objects in the namespace so the policy can be applied to related objects.

SPBM Data Objects

SPBM data objects are numerous, and for replication can be complex, but the inheritance hierarchy is relatively flat. Some important objects are explained below

`CapabilityMetadata` represents the metadata for a single unique setting as defined by the VASA provider. A simple setting has one property, while a complex setting has more than one property.

`CapabilityMetadata` contains the `CapabilityId` (below), name and description of the capability, whether it is mandatory when creating a profile, whether the capability should affect placement and compliance, a key ID for the property, whether multiple constraints are allowed for a capability instance, and metadata for properties that comprise the capability. All contents but the first two and last are optional.

`CapabilityId` contains the unique identifier for a capability within its namespace, and the namespace to which the capability belongs.

`CapabilityInstance` contains the `CapabilityId` (above), an array of constraints on the properties that comprise the capability, and (as of VASA 3.0) the appropriate line of service for the capability.

`ConstraintInstance` contains the property constraints (in `PropertyInstance`) for a single occurrence of this capability. All properties must satisfy their respective constraints to be compliant. `PropertyInstance` includes the unique identifier for the property, and a value for its constraint.

The VASA provider is responsible for sending XML descriptions of capability metadata back to SMS. In the XML capability profiles, the <`subProfiles`> have a <`name`>, followed by a <`capability`> list, with each capability having an <`id`> for the <`namespace`> and a <`constraint`> for the <`propertyInstance`>.

`CapabilityMetadataPerCategory` contains an array of `CapabilityMetadata` (above) and the category to which the metadata belongs. The category is vendor-supplied; well-known terms are recommended.

`CapabilitySchema` describes a capability namespace, as reported by the VASA Provider in response to the `queryCapabilityMetadata` call. `CapabilitySchema` contains vendor information, namespace information, an array of `CapabilityMetadataPerCategory` (above), the line of service, and a schema ID string.

vSAN Based Storage Profiles

4

Storage requirements are based on storage capabilities available from storage provider, and vSphere offers vSAN storage capabilities. To create a requirements profile for vSAN capabilities, you retrieve metadata that describes vSAN capabilities and create a subprofile that expresses the storage requirements for virtual machine or virtual disk files. To perform these operations, you connect to the Storage Policy Server

Read the following topics next:

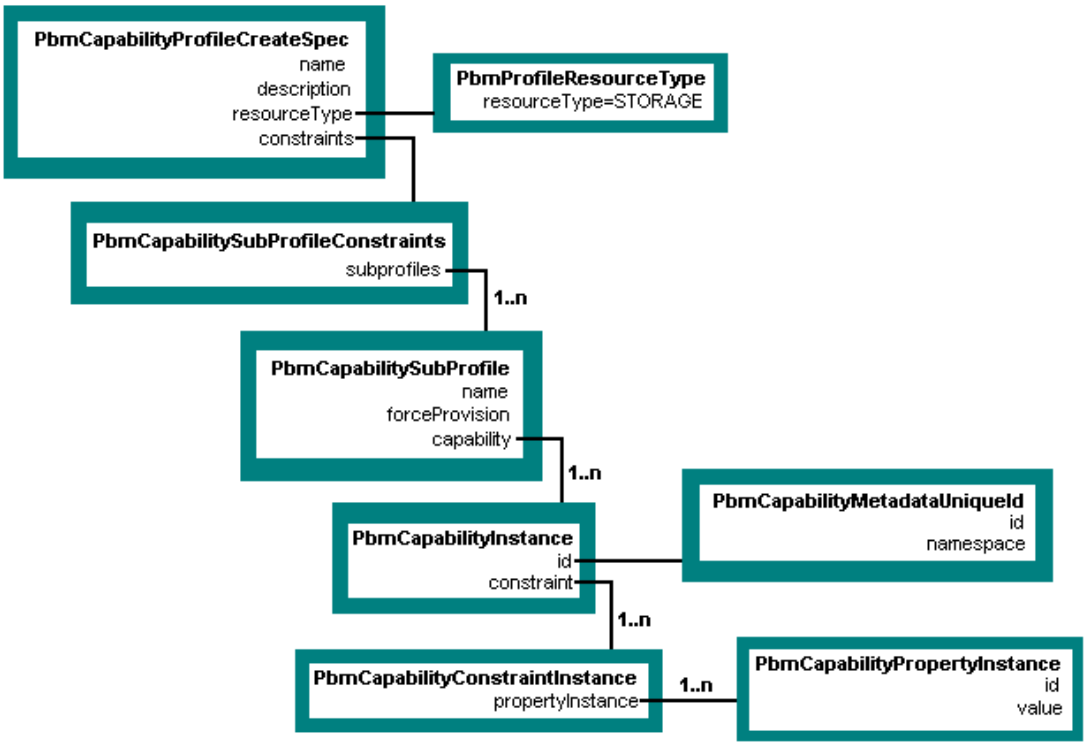
- Create a vSAN Requirements Profile
- Create an Individual Storage Requirement
- Create a Storage Profile

Create a vSAN Requirements Profile

The following example demonstrates how to create a storage requirements profile based on vSphere vSAN storage capabilities. The example creates a requirement profile for vSAN stripe width.

The following figure shows the data objects used for a profile specification.

Figure 4-1. Storage Profile Specification



The following example is based on the Storage Policy SDK sample file `CreateVSANProfile.java`. This example is divided into two code fragments.

- Create an Individual Storage Requirement – The code fragment is a function that creates a single storage capability instance for a subprofile (rule).
- Create a Storage Profile – The code fragment builds a profile specification and creates the profile.

Create an Individual Storage Requirement

The following example builds a property instance for a capability. The property instance represents a single storage requirement.

The code performs the following steps.

1 Verifies that the capability exists.

2 Creates a property instance for the requirement (`PbmCapabilityPropertyInstance`).

3 Creates a capability constraint for the property instance (`PbmCapabilityConstraintInstance`).

4 Create a capability instance for the constraint and add the subprofile (rule) to the capability.

```
PbmCapabilityInstance buildCapability(String capabilityName, Object value,
         List<PbmCapabilityMetadataPerCategory> metadata)
```

```
          throws InvalidArgumentFaultMsg {

// Retrieve the metadata for the capability (stripeWidth)
PbmCapabilityMetadata capabilityMeta = PbmUtil.getCapabilityMeta(capabilityName, metadata);
if (capabilityMeta == null)
throw new InvalidArgumentFaultMsg("Specified Capability does not exist", null);

// Create a New Property Instance based on the Stripe Width Capability
PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();
prop.setId(capabilityName);
prop.setValue(value);

// Associate Property Instance with a Rule (subprofile)
PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();
rule.getPropertyInstance().add(prop);

// Associate Rule (subprofile) with a Capability Instance
PbmCapabilityInstance capability = new PbmCapabilityInstance();
capability.setId(capabilityMeta.getId());
capability.getConstraint().add(rule);

return capability;
}
```

Create a Storage Profile

The example performs the following operations.

1 Retrieve a reference to the Storage Policy Profile Manger.

2 Verify that there is vSAN Storage Policy support.

3 Retrieve the vSAN storage capability metadata.

4 Add capabilities to be used as requirements.

5 Add the requirement capabilities to a subprofile. A subprofile corresponds to a rule set in the vSphere Web Client.

6 Specify the subprofile as capability constraints.

7 Build a profile specification.

8 Create the storage profile.

When you create a storage profile, the `PbmCreate` method returns a profile ID (`PbmProfileId`). The Profile Manager maintains a list of profiles. To obtain a profile from the list, use the `PbmQueryProfile` and `PbmRetrieveContent` methods. See Retrieve an Existing Storage Profile from the Server.

Example: vSAN Storage Profile Creation

```
// 1: Get PBM Profile Manager & Associated Capability Metadata
spbmsc = connection.getPbmServiceContent();
```

```
ManagedObjectReference profileMgr = spbmsc.getProfileManager();

// 2: Verify that there is vSAN Storage Policy support
Boolean vSanCapabale = false;
List<PbmCapabilityVendorResourceTypeInfo> vendorInfo =
     connection.getPbmPort().pbmFetchVendorInfo(profileMgr, null);
for (PbmCapabilityVendorResourceTypeInfo vendor : vendorInfo)
     for (PbmCapabilityVendorNamespaceInfo vnsi : vendor.getVendorNamespaceInfo())
          if (vnsi.getNamespaceInfo().getNamespace().equals("vSan")) {
              vSanCapabale = true;
              break;
          }

if (!vSanCapabale)
     throw new RuntimeFaultFaultMsg(
"Cannot create storage profile. vSAN Provider not found.", null);

// 3: Get PBM Supported Capability Metadata
List<PbmCapabilityMetadataPerCategory> metadata =
     connection.getPbmPort().pbmFetchCapabilityMetadata(profileMgr,
             PbmUtil.getStorageResourceType(), null);
// 4: Add Provider Specific Capabilities
List<PbmCapabilityInstance> capabilities = new ArrayList<PbmCapabilityInstance>();
capabilities.add(buildCapability("stripeWidth", stripeWidth, metadata));

// 5: Add Capabilities to a RuleSet (subprofile)
PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();
ruleSet.getCapability().addAll(capabilities);

// 6: Add Rule-Set (subprofile) to Capability Constraints
PbmCapabilitySubProfileConstraints constraints = new PbmCapabilitySubProfileConstraints();
ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));
constraints.getSubProfiles().add(ruleSet);

// 7: Build Capability-Based Profile
PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();
spec.setName(profileName);
spec.setDescription("Storage Profile Created by SDK Samples. Rule based on vSAN capability");
spec.setResourceType(PbmUtil.getStorageResourceType());
spec.setConstraints(constraints);

// 8: Create Storage Profile
PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);
System.out.println("Profile " + profileName + " created with ID: " + profile.getUniqueId());
```

Virtual Machine Storage Profiles

5

The Storage Policy Server maintains a list of storage profiles. To apply them, you retrieve an existing storage profile from the storage policy server, then associate it with a virtual machine. Code fragments in this chapter are based on sample program `VMCreate.java` in the Storage Policy SDK.

Read the following topics next:

- Retrieve an Existing Storage Profile from the Server
- Apply the Storage Profile to a Virtual Machine

Retrieve an Existing Storage Profile from the Server

You can use a script to retrieve an existing storage profile from the storage policy server.

The following code fragment shows the example function `getPbmProfileSpec` that uses the `PbmQueryProfile` and `PbmRetrieveContent` methods to retrieve storage profiles. In the context of the Storage Policy SDK example `VMCreate.java`, the function returns a `VirtualMachineDefinedProfileSpec` to be used to configure storage for a virtual machine.

The function performs the following operations:

1. Uses the connection to the Storage Policy Server to retrieve a reference to the Profile Manager.
2. Calls the `PbmQueryProfile` method to obtain the list of storage profile identifiers.
3. Calls the `PbmRetrieveContent` method to obtain the list of storage profiles.
4. Finds the profile that matches the specified profile name.
5. Creates a `VirtualMachineDefinedProfileSpec` and assigns the identifier from the named profile to the `VirtualMachineDefinedProfileSpec`. You use the `VirtualMachineDefinedProfileSpec` when you configure the virtual machine. See Apply the Storage Profile to a Virtual Machine.

Example: Retrieving a Storage Profile

```
VirtualMachineDefinedProfileSpec getPbmProfileSpec(String name)
        throws InvalidArgumentFaultMsg, com.vmware.pbm.RuntimeFaultFaultMsg,
```

```
             RuntimeFaultFaultMsg {

// 1 Get PBM Profile Manager
        PbmServiceInstanceContent spbmsc = connection.getPbmServiceContent();
        ManagedObjectReference profileMgr = spbmsc.getProfileManager();

// 2 Retrieve the list of profile identifiers.
        List<PbmProfileId> profileIds =
            connection.getPbmPort().pbmQueryProfile(profileMgr,
                                                    PbmUtil.getStorageResourceType(),
                                                    null);

        if (profileIds == null || profileIds.isEmpty())
           throw new RuntimeFaultFaultMsg("No storage Profiles exist.", null);

// 3 Retrieve the list of storage profiles.
        List<PbmProfile> pbmProfiles =
            connection.getPbmPort().pbmRetrieveContent(profileMgr, profileIds);

// 4,5 Find the named profile and create a VirtualMachineDefinedProfileSpec
//    that will use the same profile identifier.
        for (PbmProfile pbmProfile : pbmProfiles) {
            if (pbmProfile.getName().equals(name)) {
                PbmCapabilityProfile profile = (PbmCapabilityProfile) pbmProfile;
                VirtualMachineDefinedProfileSpec spbmProfile =
                    new VirtualMachineDefinedProfileSpec();

                spbmProfile.setProfileId(profile.getProfileId().getUniqueId());

                return spbmProfile;
            }
        }

// Throw exception if none found
        throw new InvalidArgumentFaultMsg(
                "Specified storage profile name does not exist.", null);
    }
```

Apply the Storage Profile to a Virtual Machine

To use a storage profile for a virtual machine, specify a `VirtualMachineDefinedProfileSpec` object for the `VirtualMachineConfigSpec.vmProfile` property.

The following code fragment sets the storage profile and creates the virtual machine. The profile (`spbmProfile`) is a `VirtualMachineDefinedProfileSpec`. See Retrieve an Existing Storage Profile from the Server.

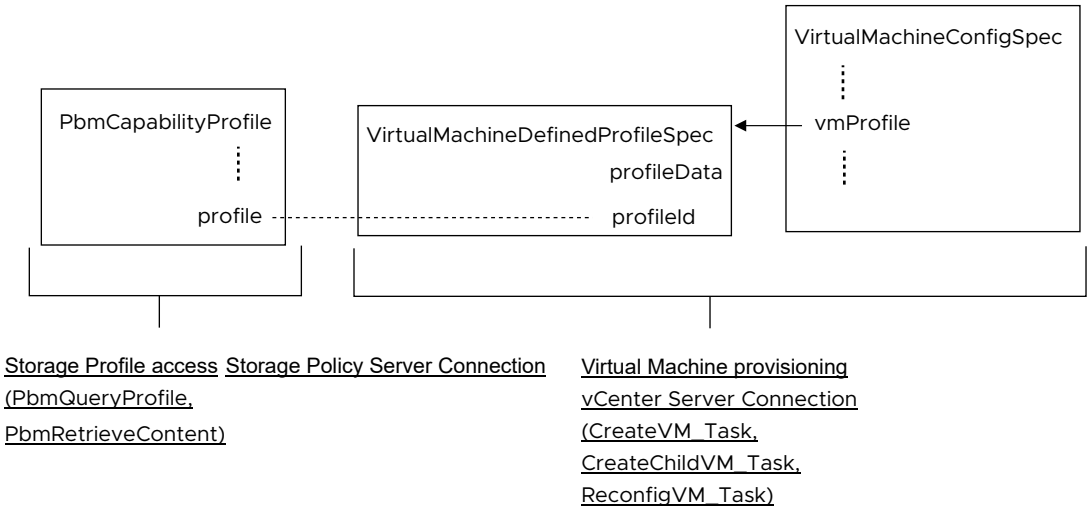
Example: Associating a Storage Profile with a Virtual Machine

```
[...]
VirtualMachineConfigSpec configSpec = new VirtualMachineConfigSpec();
// Set SPBM profile
configSpec.getVmProfile().add(spbmProfile);
```

```
[...]
ManagedObjectReference taskmor =
     connection.getVimPort().createVMTask(vmFolderMor, vmConfigSpec, resourcepoolmor,
hostmor);
```

The following figure shows how a storage profile is integrated into a virtual machine configuration specification. Your client establishes the link between the storage profile (`PbmCapabilityProfile`) and the `VirtualMachineDefinedProfileSpec` by setting the `profileId` property in the `VirtualMachineDefinedProfileSpec`. The Server sets the `profileData` property when it configures the virtual machine.

Figure 5-1. Using a Storage Profile for Virtual Machine Provisioning



Tag-Based Storage Profiles

6

To use a tag-based storage profile, you assign a storage policy tag to a data center, define a storage requirement profile based on the tag, and associate the profile with a virtual machine.

Read the following topics next:

- Creating a Tag-Based Storage Profile
- Creating a Storage Profile
- Retrieving Tag Metadata

Creating a Tag-Based Storage Profile

A storage profile specification has associated tag metadata, which you can use to create a storage profile.

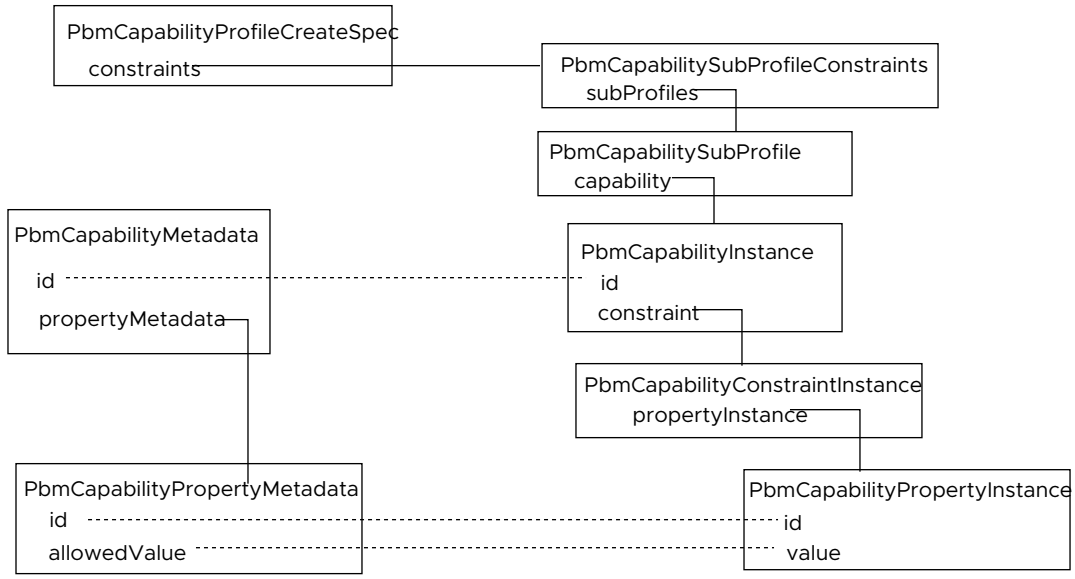
When you create the virtual machine, the vCenter Server instance will use the Storage Policy Server to resolve the tag reference in the profile and determine a datastore for virtual machine storage.

- To associate a storage policy tag with a datacenter, use the vSphere Web Client.
- To create a tag-based storage requirements profile, you retrieve metadata associated with a storage policy tag and create a storage profile that contains identifiers from the tag metadata.
- To associate the storage profile with a virtual machine, see Apply the Storage Profile to a Virtual Machine.

A tag-based storage profile involves the following metadata:

- The subprofile capability instance identifier (`PbmCapabilityInstance.id`) is set to the storage policy tag metadata identifier (`PbmCapabilityMetadata.id`).
- The capability property instance (`PbmCapabilityPropertyInstance`) specifies both an identifier and a value. Both properties are set to the tag metadata `id` and `allowedValue` properties.

Figure 6-1. Tag-Based Storage Profile Specification



The following example demonstrates how to create a storage requirements profile based on a storage policy tag. The example is divided into two sections.

- Retrieving Tag Metadata
- Creating a Storage Profile

Creating a Storage Profile

You can create a storage profile based on tag metadata.

The example performs the following operations.

1 Create a property instance with tags from the specified tag category.

2 Associate the property instance with a constraint (rule).

3 Associate the constraint with a capability instance.

4 Add the capability instance to a subprofile (rule set).

5 Add the subprofile to the list of subprofile constraints.

6 Build a profile specification.

7 Create the storage profile.

The following example is based on the Storage Policy SDK sample file `CreateProfile.java`.

Example: Tag-Based Storage Profile Creation

```
// Get PBM Profile Manager and PBM Capability Metadata
spbmsc = connection.getPbmServiceContent();
ManagedObjectReference profileMgr = spbmsc.getProfileManager();
```

```
List<PbmCapabilityMetadataPerCategory> metadata =
     connection.getPbmPort().pbmFetchCapabilityMetadata(
profileMgr,PbmUtil.getStorageResourceType(), null);

// Step 1: Create Property Instance with tags from the specified Category
PbmCapabilityMetadata tagCategoryInfo = PbmUtil.getTagCategoryMeta(tagCategoryName, metadata);

// Fetch Property Metadata of the Tag Category
List<PbmCapabilityPropertyMetadata> propMetaList = tagCategoryInfo.getPropertyMetadata();
PbmCapabilityPropertyMetadata propMeta = propMetaList.get(0);

// Create a New Property Instance based on the Tag Category ID
PbmCapabilityPropertyInstance prop = new PbmCapabilityPropertyInstance();
prop.setId(propMeta.getId());

// Fetch Allowed Tag Values Metadata; cast the xsd:any property (allowedValue) to a discrete
set
PbmCapabilityDiscreteSet tagSetMeta = (PbmCapabilityDiscreteSet) propMeta.getAllowedValue();

// Create a New Discrete Set for holding Tag Values
PbmCapabilityDiscreteSet tagSet = new PbmCapabilityDiscreteSet();
for (Object obj : tagSetMeta.getValues()) {
   tagSet.getValues().add(((PbmCapabilityDescription) obj).getValue());
}
prop.setValue(tagSet);

// Step 2: Associate Property Instance with a Rule
PbmCapabilityConstraintInstance rule = new PbmCapabilityConstraintInstance();
rule.getPropertyInstance().add(prop);

// Step 3: Associate Rule with a Capability Instance
PbmCapabilityInstance capability = new PbmCapabilityInstance();
capability.setId(tagCategoryInfo.getId());
capability.getConstraint().add(rule);

// Step 4: Add Rule to a RuleSet
PbmCapabilitySubProfile ruleSet = new PbmCapabilitySubProfile();
ruleSet.getCapability().add(capability);

// Step 5: Add Rule-Set to Capability Constraints
PbmCapabilitySubProfileConstraints constraints = new PbmCapabilitySubProfileConstraints();
ruleSet.setName("Rule-Set " + (constraints.getSubProfiles().size() + 1));
constraints.getSubProfiles().add(ruleSet);

// Step 6: Build Capability-Based Profile
PbmCapabilityProfileCreateSpec spec = new PbmCapabilityProfileCreateSpec();
spec.setName(profileName);
spec.setDescription("Tag Based Storage Profile Created by SDK Samples. Rule based on tags
from Category "
            + tagCategoryName);
spec.setResourceType(PbmUtil.getStorageResourceType());
spec.setConstraints(constraints);
```

```
// Step 7: Create Storage Profile
PbmProfileId profile = connection.getPbmPort().pbmCreate(profileMgr, spec);
```

Retrieving Tag Metadata

You can retrieve tag metadata for a tag category.

The following example shows a code fragment that retrieves metadata for a tag category. Given the list of metadata obtained from the Storage Policy Server, the function traverses the list and returns the metadata associated with the specified category. This function is defined in the `PbmUtil` package in the Storage Policy SDK.

Example: getTagCategoryMeta (PbmUtil Package)

```
public static PbmCapabilityMetadata getTagCategoryMeta(
     String tagCategoryName, List<PbmCapabilityMetadataPerCategory> schema) {
          for (PbmCapabilityMetadataPerCategory cat : schema)
              if (cat.getSubCategory().equals("tag"))
                  for (PbmCapabilityMetadata cap : cat.getCapabilityMetadata())
                      if (cap.getId().getId().equals(tagCategoryName))
                         return cap;
          return null;
   }
```

Policy Rules 7

You can create custom storage policy strings to specify policy rules.

Read the following topics next:

- Create a Stand-Alone Virtual Disk with an Attached Storage Encryption Policy

Create a Stand-Alone Virtual Disk with an Attached Storage Encryption Policy

The following example shows how to format an XML string to specify an I/O filter policy for a virtual disk.

The format is specific to the current version of the VMware Web Services API. This format might not be compatible with other API versions.

Prerequisites

- Verify that the name for the policy is user-friendly.
- Verify that the ID string to identify the policy is unique. The ID must be unique among the set of policies known to SPBM.

Procedure

1 Build the policy `<capability>` element, which specifies the filter.

```
<capability>
   <capabilityId>
      <id>vmwarevmcrypt@encryption</id>
      <namespace>IOFILTERS</namespace>
      <constraint></constraint>
   </capabilityId>
</capability>
```

2 Wrap the `<capability>` element singly in a `<subProfiles>` element, adding the rule name.

```
<subProfiles>
   <capability>
      <capabilityId>
         <id>vmwarevmcrypt@encryption</id>
         <namespace>IOFILTERS</namespace>
```

```
          <constraint></constraint>
       </capabilityId>
    </capability>
    <name>Rule-Set 1: IOFILTERS</name>
</subProfiles>
```

3 Wrap the <subProfiles> element singly in a <constraints> element.

```
<constraints>
  <subProfiles>
    <capability>
       <capabilityId>
          <id>vmwarevmcrypt@encryption</id>
          <namespace>IOFILTERS</namespace>
          <constraint></constraint>
       </capabilityId>
    </capability>
    <name>Rule-Set 1: IOFILTERS</name>
  </subProfiles>
</constraints>
```

4 Append profile metadata, such as name and creation date.

```
<constraints>
  <subProfiles>
    <capability>
      <capabilityId>
        <id>vmwarevmcrypt@encryption</id>
        <namespace>IOFILTERS</namespace>
        <constraint></constraint>
      </capabilityId>
    </capability>
    <name>Rule-Set 1: IOFILTERS</name>
  </subProfiles>
</constraints>
<createdBy>client</createdBy>
<creationTime>1999-12-31T23:59:59Z</creationTime>
<lastUpdatedTime>1999-12-31T23:59:59Z</lastUpdatedTime>
<generationId>1</generationId>
<name>IOfilter-Encrypt</name>
```

5 Wrap the <constraints> element and metadata in a <storageProfile> element, and prefix an XML header.

```
<?xml version='1.0' encoding='UTF-8'?>
<storageProfile xsi:type='StorageProfile'>
  <constraints>
    <subProfiles>
      <capability>
        <capabilityId>
          <id>vmwarevmcrypt@encryption</id>
          <namespace>IOFILTERS</namespace>
          <constraint></constraint>
        </capabilityId>
```

```
      </capability>
      <name>Rule-Set 1: IOFILTERS</name>
    </subProfiles>
  </constraints>
  <createdBy>client</createdBy>
  <creationTime>1999-12-31T23:59:59Z</creationTime>
  <lastUpdatedTime>1999-12-31T23:59:59Z</lastUpdatedTime>
  <generationId>1</generationId>
  <name>IOfilter-Encrypt</name>
  <profileId>I am Unique</profileId>
</storageProfile>
```

What to do next

You can create a `ProfileSpec` to contain the XML data. See the *vSphere Web Services SDK Programming Guide*.

Legacy Storage Profiles

8

vSphere 5.0/5.1 systems support limited storage capability and requirement profiles.

In vSphere 5.5 and later, the Storage Policy Server supports more complex storage profiles. vSphere 5.5 and later upgrade legacy capability and requirement profiles for Storage Policy Server operations.

Read the following topics next:

- VASA 1.0 Storage Capability Upgrade
- vSphere Web Client User Label Conversion

VASA 1.0 Storage Capability Upgrade

A Storage Policy Server can obtain storage capability data from VASA providers.

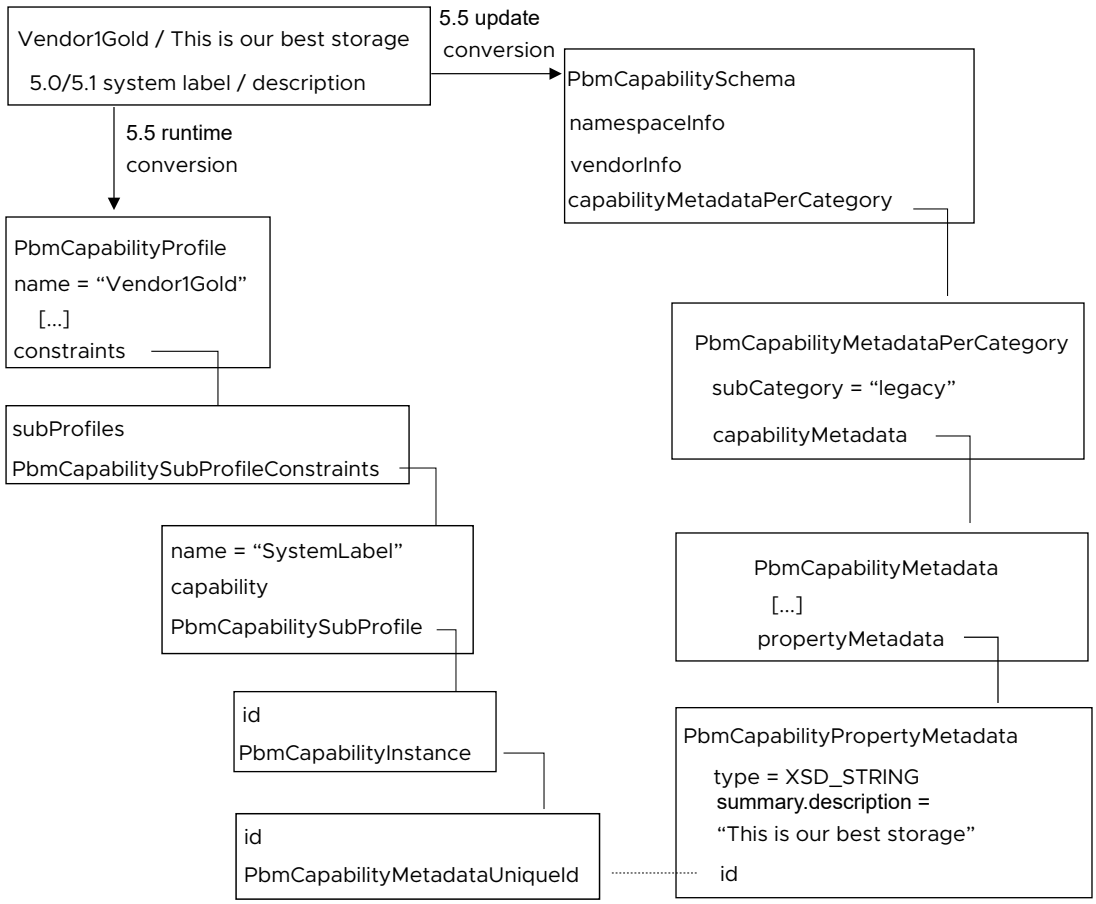
In vSphere 5.5, this generally implies VMware vSAN storage capabilities. A Storage Policy Server can also obtain capability data from a VASA provider that was implemented for the vSphere 5.0/5.1 environment.

The early architecture (vSphere 5.0/5.1) supports a simple expression of storage capability. A VASA 1.0 provider can advertise one system label per datastore. A system label has an associated description.

The Storage Policy Server performs a runtime conversion of VASA 1.0 system labels. The Storage Policy API presents the system label as a storage capability profile. The Server also generates a capability schema for the storage label. The generated storage capability profile references the generated schema.

The following figure shows the Storage Policy data objects that are generated from a vSphere 5.0/5.1 legacy profile.

Figure 8-1. Converted Legacy Capability Profile



vSphere Web Client User Label Conversion

A vSphere 5.0/5.1 storage profile can reference user labels displayed in the vSphere Web Client.

Users can associate a datastore with a user label. When you upgrade to vSphere 5.5, any existing 5.0/5.1 user labels will be converted into datastore tags.

A converted policy profile contains one subprofile for each label referenced by the original vSphere 5.0/5.1 profile.

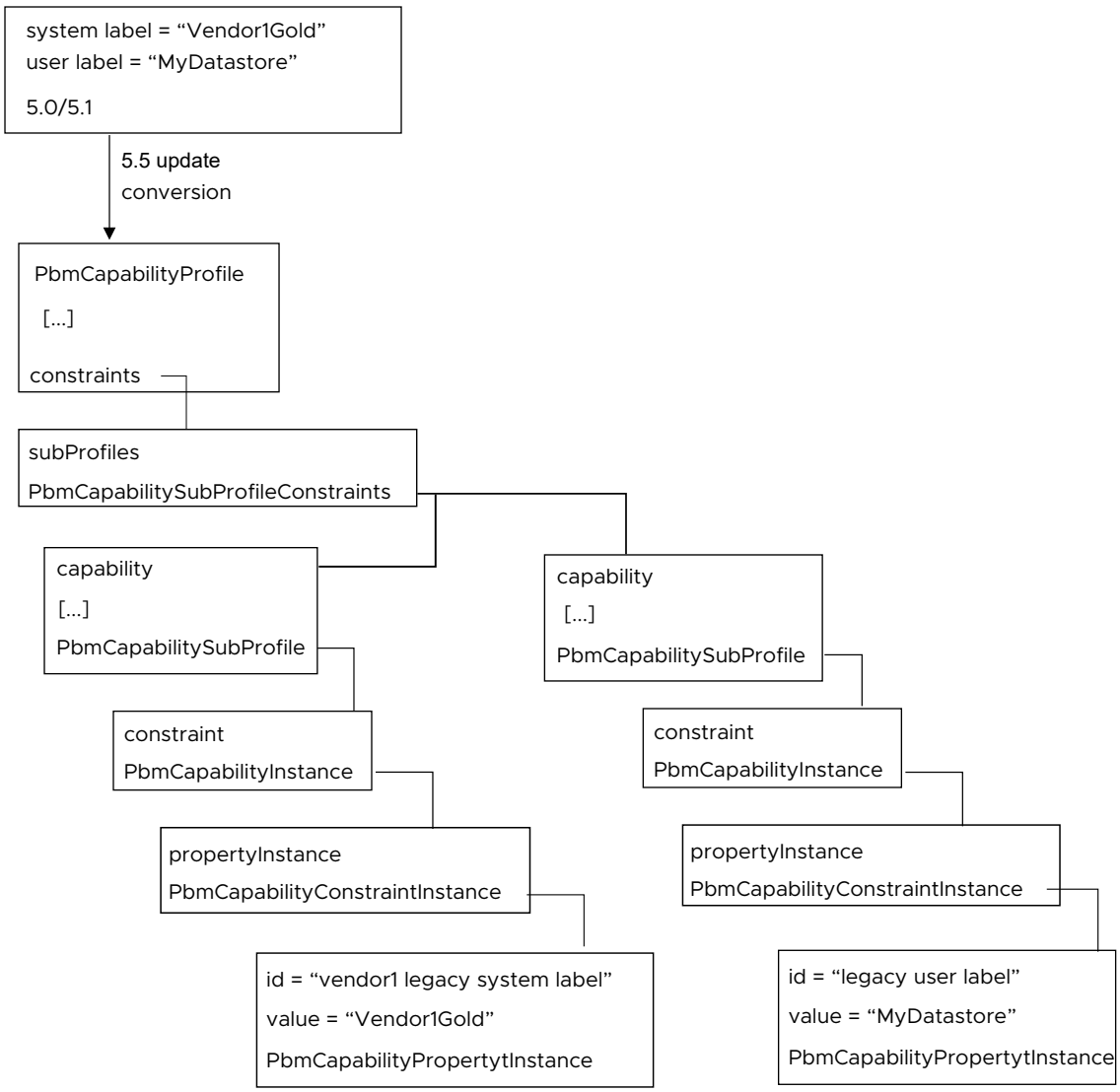
- A system label reference is converted to a reference to the appropriate vendor-specific "legacy system label" capability generated from that label.
- A user label reference is converted to a reference to the appropriate datastore tag generated from that label.

The following figure shows the conversion of a vSphere 5.0/5.1 profile that references a system label "Vendor1Gold" and a user label "MyDatastores". When you upgrade to vSphere 5.5, the system converts the profile into Storage Policy API elements.

- The legacy system label identifies the original source as a VASA 1.0 provider.

- The legacy user label identifies the original source as a 5.0/5.1 user label in the vSphere Web Client.

Figure 8-2. Converted Legacy Requirement Profile



vCenter Single Sign-On Client Example

9

This chapter describes a Java example of acquiring a vCenter Single Sign-On security token.

Read the following topics next:

- vCenter Single Sign-On Token Request Overview
- Using Handler Methods for SOAP Headers
- Sending a Request for a Security Token

vCenter Single Sign-On Token Request Overview

The code examples in the following sections show how to use the `Issue` method to acquire a holder-of-key security token.

To see an example of using the token to login to a vCenter Server, see Chapter 10 vCenter LoginByToken Example. The code examples in this chapter are based on the following sample file located in the vCenter Single Sign-On SDK JAX-WS client `samples` directory.

`.../JAXWS/samples/com/vmware/sso/client/samples/AcquireHoKTokenByUserCredentialSample.java`

The `AcquireHoKTokenByUserCredentialSample` program creates a token request and calls the issue method to send the request to a vCenter Single Sign-On Server. The program uses a sample implementation of Web services message handlers to modify the SOAP security header for the request message.

This example uses the username-password security policy (`STSSecPolicy_UserPwd`). This policy requires that the SOAP security header include a timestamp, username and password, and a digital signature and certificate. The sample message handlers embed these elements in the message.

The example performs the following operations.

1. Create a security token service client object (`STSService_Service`). This object manages the vCenter Single Sign-On header handlers and it provides access to the vCenter Single Sign-On client API methods. This example uses the issue method.
2. Create a vCenter Single Sign-On header handler resolver object (`HeaderHandlerResolver`). This object acts as a container for the different handlers.

3 Add the handlers for timestamp, user credentials, certificate, and token extraction to the handler resolver.

4 Add the handler resolver to the security token service.

5 Retrieve the STS port (`STS_Service`) from the security token service object.

6 Create a security token request.

7 Set the request fields.

8 Set the endpoint in the request context. The endpoint identifies the vCenter Single Sign-On Server.

9 Call the issue method, passing the token request.

10 Handle the response from the vCenter Single Sign-On server.

Using Handler Methods for SOAP Headers

The VMware vCenter Single Sign-On SDK provides sample code that is an extension of the JAX-WS XML Web services message handler (`javax.xml.ws.handler`).

The sample code consists of a set of SOAP header handler methods and a header handler resolver, to which you add the handler methods. The handler methods insert timestamp, user credential, and message signature data into the SOAP security header for the request. A handler method extracts the SAML token from the vCenter Single Sign-On Server response.

The VMware vCenter Single Sign-On client SOAP header handler files are located in the `soaphandlers` directory.

`SDK/sso/java/JAXWS/samples/com/vmware/sso/client/soaphandlers`

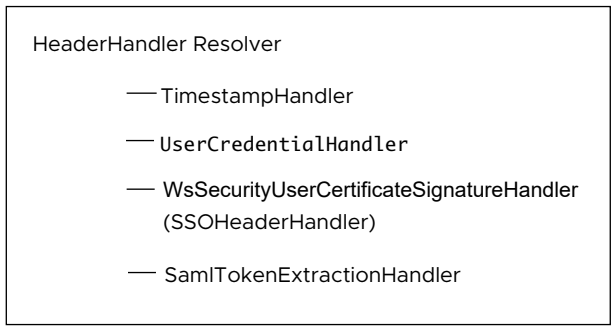
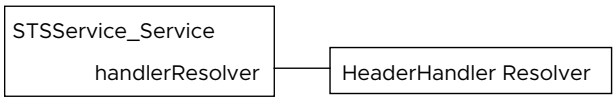
To access the SOAP handler implementation, the example code contains the following import statements.

```
import com.vmware.sso.client.soaphandlers.HeaderHandlerResolver;
import com.vmware.sso.client.soaphandlers.SSOHeaderHandler;
import com.vmware.sso.client.soaphandlers.SamlTokenExtractionHandler
import com.vmware.sso.client.soaphandlers.TimeStampHandler;
import com.vmware.sso.client.soaphandlers.UserCredentialHandler;
import com.vmware.sso.client.soaphandlers.WsSecurityUserCertificateSignatureHandler;
```

This example uses the following handler elements.

- `HeaderHandlerResolver`
- `SamlTokenExtractionHandler`
- `TimestampHandler`
- `UserCredentialHandler`
- `WsSecurityUserCertificateSignatureHandler` (`SSOHeaderHandler`)

The following sequence shows the operations and corresponding Java elements for message security.

Create an STS service object (`STSService_Service`). This object will bind the handlers to the request and provide access to the issue method.	STSService_Service
Create a handler resolver object (`HeaderHandlerResolver`). This object acts as a receptacle for the handlers.	HeaderHandlerResolver
Add the header handlers: ■ Timestamp – The handler will use system time to set the timestamp values. ■ User credential – The handler requires a username and a password; it will create a username token for the supplied values. ■ User certificate signature – The handler requires a private key and an x509 certificate. The handler will use the private key to sign the body of the SOAP message (the token request), and it will embed the certificate in the SOAP security header. ■ SAML token extraction – The handler extracts the SAML token directly from vCenter Single Sign-On Server response to avoid token modification by the JAX-WS bindings.	
Add the handler resolver to the STS service.	

The following code fragment creates a handler resolver and adds the handler methods to the handler resolver. After the handlers have been established, the client creates a token request and calls the `Issue` method. See Sending a Request for a Security Token.

Important You must perform these steps for message security before retrieving the STS service port. An example of retrieving the STS service port is shown in Sending a Request for a Security Token.

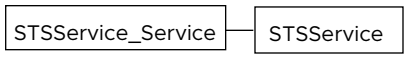
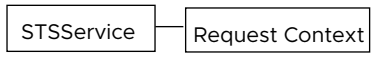
```
/*
 * Instantiate the STS Service
 */
STSService_Service stsService = new STSService_Service();

/*
 * Instantiate the HeaderHandlerResolver.
 */
HeaderHandlerResolver headerResolver = new HeaderHandlerResolver();
```

```
/*
 * Add handlers to insert a timestamp and username token into the SOAP security header
 * and sign the message.
 *
 * -- Timestamp contains the creation and expiration time for the request
 * -- UsernameToken contains the username/password
 * -- Sign the SOAP message using the combination of private key and user certificate.
 *
 * Add the TimeStampHandler
 */
headerResolver.addHandler(new TimeStampHandler());

/*
 * Add the UserCredentialHandler. arg[1] is the username; arg[2] is the password.
 */
UserCredentialHandler ucHandler = new UserCredentialHandler(args[1],args[2]);
headerResolver.addHandler(ucHandler);

/*
 * Add the message signature handler (WsSecurityUserCertificateSignatureHandler);
 * The client is responsible for supplying the private key and certificate.
 */
SSOHeaderHandler ssoHandler = new WsSecurityUserCertificateSignatureHandler(privateKey,
userCert);
headerResolver.addHandler(ssoHandler);

/*
 * Add the token extraction handler (SamlTokenExtractionHandler).
 */
SamlTokenExtractionHandler sbHandler = new SamlTokenExtractionHandler;
headerResolver.addHandler(sbHandler);

/*
 * Set the handlerResolver for the STSService to the HeaderHandlerResolver created above.
 */
stsService.setHandlerResolver(headerResolver);
```

Sending a Request for a Security Token

After setting up the SOAP header handlers, the example creates a token request and calls the `Issue` method.

The following sequence shows the operations and corresponding Java elements.

Retrieve the STS service port (`STSService`). The service port provides access to the vCenter Single Sign-On client API methods. The vCenter Single Sign-On handler resolver must be associated with the STS service before you retrieve the service port. See Using Handler Methods for SOAP Headers.	
Create a token request (`RequestSecurityTokenType`). Your vCenter Single Sign-On client will pass the token request to the `Issue` method. The `Issue` method will send the token request in the body of the SOAP message. This example sets the token request fields as appropriate for a holder-of-key token request.	RequestSecurityTokenType
Set the token request fields. ■ lifetime – Creation and expiration times. ■ token type – urn:oasis:names:tc:SAML:2.0:assertion ■ request type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue ■ key type – http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey (for holder-of-key token type) ■ signature algorithm – http://www.w3.org/2001/04/xmldsig-more#rsa-sha256 ■ renewable status	RequestSecurityTokenType tokenType requestType lifetime keyType signatureAlgorithm renewing
Set the endpoint address for the token request.	
Call the `Issue` method.	Issue (RequestSecurityTokenType) STSService
Handle the response from the vCenter Single Sign-On Server.	RequestSecurityTokenResponseType

The following example shows Java code that performs these operations.

Example: Acquiring a vCenter Single Sign-On Token – Sending the Request

```
/*
 * Retrieve the STSServicePort from the STSService_Service object.
 */
STSService stsPort = stsService.getSTSServicePort();

/*
 * Create a token request object.
 */
RequestSecurityTokenType tokenType = new RequestSecurityTokenType();

/*
 *  Create a LifetimeType object.
 */
LifetimeType lifetime = new LifetimeType();

/*
 * Derive the token creation date and time.
```

```
 * Use a GregorianCalendar to establish the current time,
 * then use a DatatypeFactory to map the time data to XML.
 */
DatatypeFactory dtFactory = DatatypeFactory.newInstance();
GregorianCalendar cal = new GregorianCalendar(TimeZone.getTimeZone("GMT"));
XMLGregorianCalendar xmlCalendar = dtFactory.newXMLGregorianCalendar(cal);
AttributedDateTime created = new AttributedDateTime();
created.setValue(xmlCalendar.toXMLFormat());

/*
 * Specify a time interval for token expiration (specified in milliseconds).
 */
AttributedDateTime expires = new AttributedDateTime();
xmlCalendar.add(dtFactory.newDuration(30 * 60 * 1000));
expires.setValue(xmlCalendar.toXMLFormat());

/*
 * Set the created and expires fields in the lifetime object.
 */
lifetime.setCreated(created);
lifetime.setExpires(expires);

/*
 * Set the token request fields.
 */
tokenType.setTokenType("urn:oasis:names:tc:SAML:2.0:assertion");
tokenType.setRequestType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue");
tokenType.setLifetime(lifetime);
tokenType.setKeyType("http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey");
tokenType.setSignatureAlgorithm("http://www.w3.org/2001/04/xmldsig-more#rsa-sha256");

/*
 * Specify a token that can be renewed.
 */
RenewingType renewing = new RenewingType();
renewing.setAllow(Boolean.TRUE);
renewing.setOK(Boolean.FALSE); // WS-Trust Profile: MUST be set to false
tokenType.setRenewing(renewing);

/* Get the request context and set the endpoint address. */
Map<String, Object> reqContext = ((BindingProvider) stsPort).getRequestContext();
reqContext.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, args[0]);

/*
 * Use the STS port to invoke the "issue" method to acquire the token
 * from the vCenter Single Sign-On Server.
 */
RequestSecurityTokenResponseCollectionType issueResponse = stsPort.issue(tokenType);

/*
 * Handle the response - extract the SAML token from the response. The response type
 * contains the token type (SAML token type urn:oasis:names:tc:SAML:2.0:assertion).
 */
RequestSecurityTokenResponseType rstResponse =
issueResponse.getRequestSecurityTokenResponse();
```

```
RequestedSecurityTokenType requestedSecurityToken = rstResponse.getRequestedSecurityToken();

/*
 * Extract the SAML token from the RequestedSecurityTokenType object.
 * The generic token type (Element) corresponds to the type required
 * for the SAML token handler that supports the call to LoginByToken.
 */
Element token = requestedSecurityToken.getAny();
```

vCenter LoginByToken Example

10

This chapter describes a Java example of using the `LoginByToken` method.

Read the following topics next:

- vCenter Server Single Sign-On Session
- Saving the vCenter Server Session Cookie
- Using LoginByToken
- Restoring the vCenter Server Session Cookie

vCenter Server Single Sign-On Session

After you obtain a SAML token from the vCenter Single Sign-On Server, you can use the vSphere API method `LoginByToken` to establish a single sign-on session with a vCenter Server instance.

See Chapter 9 vCenter Single Sign-On Client Example for an example of obtaining a vCenter Single Sign-On token.

At the beginning of a vCenter Single Sign-On session, your client is responsible for the following tasks:

- Maintain the vCenter session cookie. The vSphere architecture uses an HTTP cookie to support a persistent connection between a vSphere client and a vCenter Server instance. During the initial connection, the Server produces a session cookie. Operations during the login sequence will reset the request context so your client must save this cookie and re-introduce it at the appropriate times.
- Insert the vCenter Single Sign-On token and a timestamp into the SOAP header of the `LoginByToken` message.

The example program uses these general steps.

1 Call the `RetrieveServiceContent` method to establish an HTTP connection with the vCenter Server instance and save the HTTP session cookie. The client uses an HTTP header handler method to extract the cookie from the vCenter Server response.

2 Call the `LoginByToken` method to authenticate the vCenter session. To send the token to the vCenter Server instance, the client uses a handler to embed the token and a time stamp in the SOAP header for the message. To identify the session started with the `RetrieveServiceContent` method, the client uses a handler to embed the session cookie in the HTTP header.

3 Restore the session cookie.

HTTP and SOAP Header Handlers

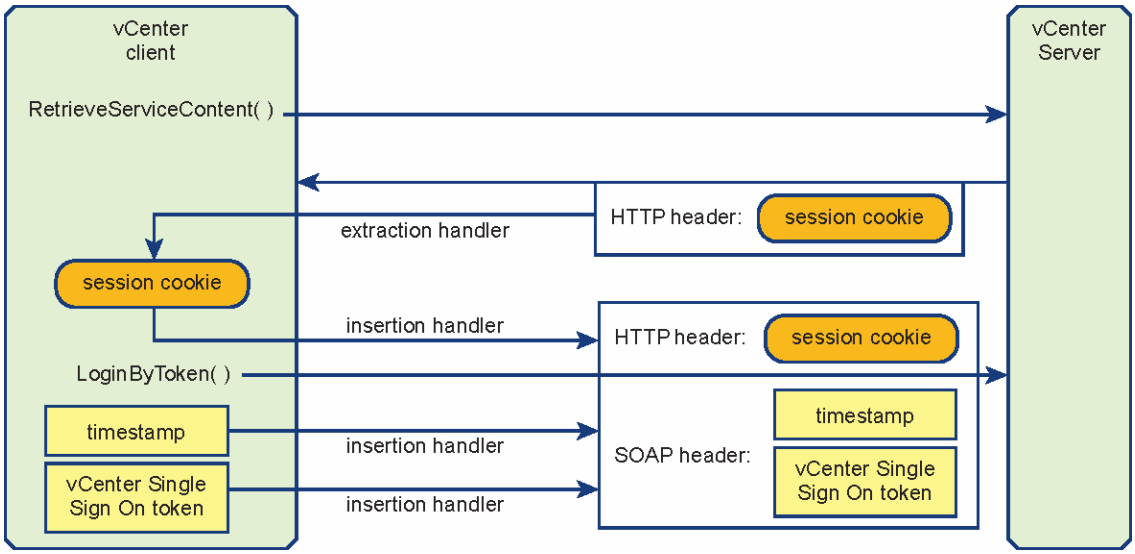
To use a vCenter Single Sign-On token to login to a vCenter Server instance, the example uses header handlers to manipulates the HTTP and SOAP header elements of the login request.

After establishing a handler, subsequent requests automatically invoke the handler.

- An extraction handler obtains the HTTP session cookie provided by the vCenter Server instance. After setting up the handler, a call to the `RetrieveServiceContent` method will invoke the handler to extract the cookie from the Server response.
- Insertion handlers put the vCenter Single Sign-On token and a timestamp into the SOAP header and the session cookie into the HTTP header of the login request.

The following figure shows the use of handlers to manipulate header elements when establishing a vCenter Single Sign-On session with a vCenter Server instance.

Figure 10-1. Starting a vCenter Session



Important Every call to the vCenter Server instance will invoke any message handlers that have been established. The overhead involved in using the SOAP and HTTP message handlers is not necessary after the session has been established. The example saves the default message handler before setting up the SOAP and HTTP handlers. After establishing the session, the example will reset the handler chain and restore the default handler.

The example code also uses multiple calls to the `VimPortType.getVimPort method` to manage the request context. The `getVimPort` method clears the HTTP request context. After each call to the `getVimPort` method, the client resets the request context endpoint address to the vCenter Server URL. After the client has obtained the session cookie, it will restore the cookie in subsequent requests.

LoginByToken Sample Code

The code examples in the following sections show how to use the `LoginByToken` method with a holder-of-key security token.

The code examples are based on the sample code contained in the vCenter Single Sign-On SDK. The files are located in the Java samples directory (`SDK/ssoclient/java/JAXWS/samples`).

- LoginByToken sample:

  `samples/com/vmware/vsphere/samples/LoginByTokenSample.java`

- Header cookie handlers:

  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieHandler.java`

  `samples/com/vmware/vsphere/soaphandlers/HeaderCookieExtractionHandler.java`

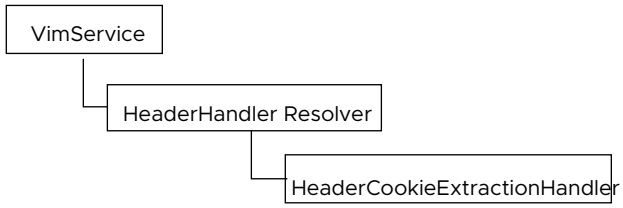
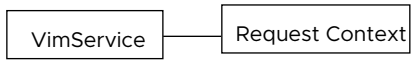
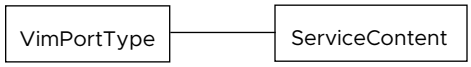
- SOAP header handlers. These are the same handlers that are used in Chapter 10 vCenter LoginByToken Example. The SOAP handler files are located in the vCenter Single Sign-On client `soaphandlers` directory:

```
samples/com/vmware/sso/client/soaphandlers
```

Saving the vCenter Server Session Cookie

The code fragment in this section establishes an HTTP session with the vCenter Server instance and saves the HTTP session cookie.

The following sequence describes these steps and shows the corresponding objects and methods.

Use the `getHandlerResolver` method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unnecessary after login.	VimService.getHandlerResolver( )
Set the cookie handler. The `HeaderCookieExtractionHandler` method retrieves the HTTP cookie.	
Get the VIM port. The VIM port provides access to the vSphere API methods, including the `LoginByToken` method.	
Set the request context endpoint address to the vCenter Server URL.	
Retrieve the `ServiceContent`. This method establishes the HTTP connection and sets the session cookie.	
Extract the cookie and save it for later use.	HeaderCookieExtractionHandler.getCookie ( )

The following example shows Java code that saves the session cookie.

Example: Saving the vCenter Server Session Cookie

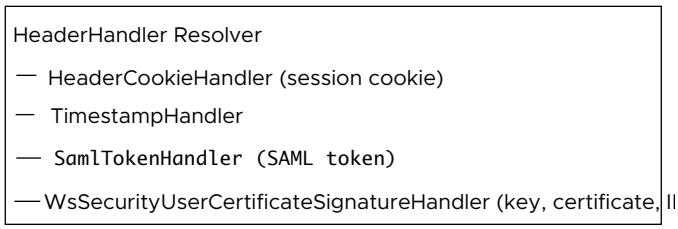
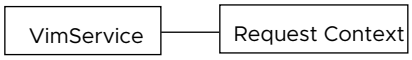
```
/*
 * The example uses a SAML token (obtained from a vCenter Single Sign-On Server)
 * and the vCenter Server URL.
 * The following declarations indicate the datatypes; the token datatype (Element) corresponds
 * to the token datatype returned by the vCenter Single Sign-On Server.
 *
 * Element token;       -- from vCenter Single Sign-On Server
 * String vcServerUrl;  -- identifies vCenter Server
 *
 * First, save the default message handler.
```

```
 */

HandlerResolver defaultHandler = vimService.getHandlerResolver();

/*
 * Create a VIM service object.
 */
vimService = new VimService();

/*
 * Construct a managed object reference for the ServiceInstance.
 */
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

/*
 * Create a handler resolver.
 * Create a cookie extraction handler and add it to the handler resolver.
 * Set the VIM service handler resolver.
 */
HeaderCookieExtractionHandler cookieExtractor = new HeaderCookieExtractionHandler();
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(cookieExtractor);
vimService.setHandlerResolver(handlerResolver);

/*
 * Get the VIM port for access to vSphere API methods. This call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Get the request context and set the connection endpoint.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Retrieve the ServiceContent. This call establishes the HTTP connection.
 */
serviceContent = vimPort.retrieveServiceContent(SVC_INST_REF);

/*
 * Save the HTTP cookie.
 */
String cookie = cookieExtractor.getCookie();
```

Using LoginByToken

The code fragment in this section sets up the message handlers and calls the `LoginByToken` method.

The following sequence describes the steps and shows the corresponding objects and methods.

Create a new `HeaderHandlerResolver`. Then set the message security handlers for cookie insertion and for inserting the SAML token and credentials in the SOAP header.	
Get the VIM port.	
Set the connection endpoint in the HTTP request context.	
Call the `LoginByToken` method. The method invocation executes the handlers to insert the elements into the message headers. The method authenticates the session referenced by the session cookie.	VimPortType.LoginByToken ( )

The following examples shows Java code that calls the `LoginByToken` method.

Example: Using LoginByToken

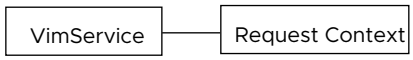
```
/*
 * Create a handler resolver and add the handlers.
 */
HeaderHandlerResolver handlerResolver = new HeaderHandlerResolver();
handlerResolver.addHandler(new TimeStampHandler());
handlerResolver.addHandler(new SamlTokenHandler(token));
handlerResolver.addHandler(new HeaderCookieHandler(cookie));
handlerResolver.addHandler(new WsSecuritySignatureAssertionHandler(
                           userCert.getPrivateKey(),
                           userCert.getUserCert(),
                           Utils.getNodeProperty(token, "ID")));
vimService.setHandlerResolver(handlerResolver);

/*
 *  Get the Vim port; this call clears the request context.
 */
vimPort = vimService.getVimPort();

/*
 * Retrieve the request context and set the server URL.
 */
Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
```

```
 * Call LoginByToken.
 */
UserSession us = vimPort.loginByToken(serviceContent.getSessionManager(), null);
```

Restoring the vCenter Server Session Cookie

After you log in, you must restore the standard vCenter session context.

The code fragment in this section restores the default message handler and the session cookie. As the cookie handler has been replaced by the default handler, the client resets the session cookie by calling request context methods to access the context fields directly. The following sequence describes these steps and shows the corresponding objects and methods.

Restore the default message handler. The handlers used for `LoginByToken` are not used in subsequent calls to the vSphere API.	VimService.setHandlerResolver ( )
Get the VIM port.	
Set the connection endpoint in the HTTP request context.	
Set the HTTP request header (vCenter session cookie).	RequestContext.get () RequestContext.put ( )

The following example shows Java code that restores the vCenter session. This code requires the vCenter URL and the cookie and default handler that were retrieved before login. See LoginByToken Sample Code.

Example: Restoring the vCenter Server Session

```
/*
 * Reset the default handler. This overwrites the existing handlers, effectively removing
them.
 */
vimService.setHandlerResolver(defaultHandler);
vimPort = vimService.getVimPort();

/*
 * Restore the connection endpoint in the request context.
 */
// Set the validated session cookie and set it in the header for once,
// JAXWS will maintain that cookie for all the subsequent requests

Map<String, Object> ctxt = ((BindingProvider) vimPort).getRequestContext();
ctxt.put(BindingProvider.ENDPOINT_ADDRESS_PROPERTY, vcServerUrl);
```

```
ctxt.put(BindingProvider.SESSION_MAINTAIN_PROPERTY, true);

/*
 * Reset the cookie in the request context.
 */
Map<String, List<String>> headers = (Map<String, List<String>>)
ctxt.get(MessageContext.HTTP_REQUEST_HEADERS);
if (headers == null) {
    headers = new HashMap<String, List<String>>();
}
headers.put("Cookie", Arrays.asList(cookie));
ctxt.put(MessageContext.HTTP_REQUEST_HEADERS, headers);
```