vCenter Single Sign On Programming Guide

vCenter Single Sign On SDK

vSphere 5.5

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-000832-02

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 This Book

vCenter Single Sign-On Programming Guide describes how to use the VMware® vCenter Single Sign On API.

VMware provides different APIs and SDKs for different applications and goals. The vCenter Single Sign On SDK supports the development of vCenter clients that use SAML token authentication for access to vSphere environments.

To view the current version of this book as well as all VMware API and SDK documentation, go to http://www.vmware.com/support/pubs/sdk_pubs.html.

Revision History

This book is revised with each release of the product or when necessary. A revised version can contain minor or major changes. Table 1 summarizes the significant changes in each version of this book.

Table 1. Revision History

Revision Date	Description
12Sep2013	vSphere 2013 release. Added documentation for C# (.NET) samples.
08Nov2012	vCenter Single Sign On SDK V1.0 documentation update – changed SOAP envelope description to identify SSL/TLS (Transport Layer Security) correctly.
10Sep2012	vCenter Single Sign On SDK V1.0 documentation.

Intended Audience

This book is intended for anyone who needs to develop applications using the vCenter Single Sign On SDK. An understanding of Web Services technology and some programming background in one of the stub languages (Java or C#) is required.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation go to http://www.vmware.com/support/pubs.

Document Feedback

VMware welcomes your suggestions for improving our documentation. Send your feedback to docfeedback@vmware.com.

Single Sign-On in the vSphere Environment 1

A vCenter Single Sign-On client connects to the vCenter Single Sign On Server to obtain a security token that contains authentication claims required for operations in the vSphere environment. The vCenter Single Sign On client API supports operations to acquire, renew, and validate tokens.

This chapter includes the following topics:

vCenter Single Sign On Overview

To support the requirements for secure software environments, software components require authorization to perform operations on behalf of a user. In a single sign-on environment, a user provides credentials once, and components in the environment perform operations based on the original authentication. vCenter Single Sign On authentication can use the following identity store technologies:

- Windows Active Directory
- OpenLDAP (Lightweight Directory Access Protocol)
- Local user accounts (vCenter Single Sign On Server resident on the vCenter Server machine)
- vCenter Single Sign On user accounts

For information about configuring identity store support, see *vSphere Installation and Setup* and *vSphere Security* in the VMware Documentation Center.

In the context of single sign-on, the vSphere environment is a collection of services and solutions, each of which potentially requires authentication of clients that use the service or solution. Examples of solutions that might support single sign-on include vShield, SRM (Site Recovery Manager), and vCO (vCenter Orchestrator). Because a service can use another service, single sign-on provides a convenient mechanism to broker authentication during a sequence of vSphere operations.

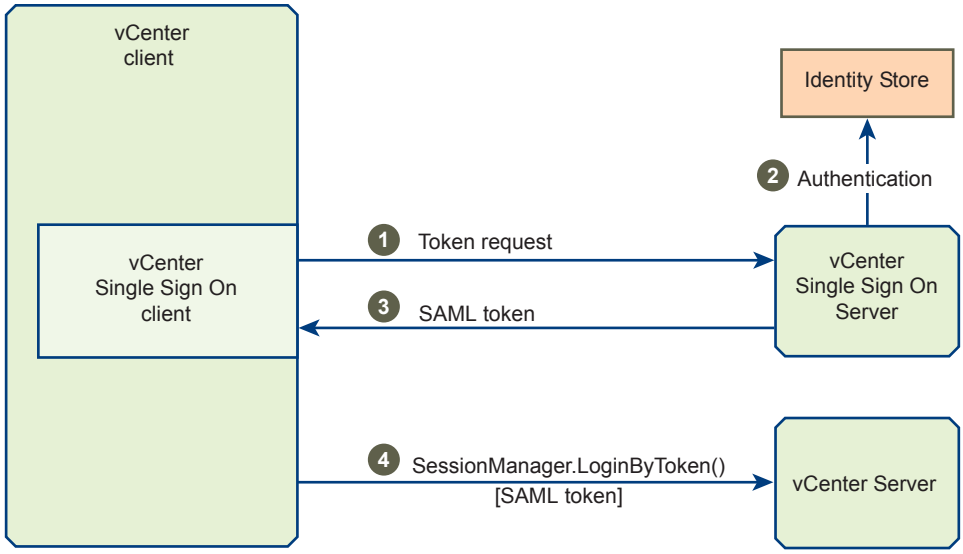
The vCenter Single Sign On Server provides a Security Token Service (STS). A vCenter Single Sign On client connects to the vCenter Single Sign On server to obtain a token that represents the client. A token uses the Security Assertion Markup Language (SAML) which is an XML encoding of authentication data. It contains a collection of statements or claims that support client authentication. Examples of token claims include name, key, and group.

There are two types of vCenter Single Sign On tokens.

- Holder-of-key tokens provide authentication based on security artifacts embedded in the token. Holder-of-key tokens can be used for delegation. A client can obtain a holder-of-key token and delegate that token for use by another entity. The token contains the claims to identify the originator and the delegate. In the vSphere environment, a vCenter Server obtains delegated tokens on a user's behalf and uses those tokens to perform operations.
- Bearer tokens provide authentication based only on possession of the token. Bearer tokens are intended for short-term, single-operation use. A bearer token does not verify the identity of the user (or entity) sending the request. It is possible to use bearer tokens in the vSphere environment, however there are potential limitations:
  - The vCenter Single Sign On Server may impose limitations on the token lifetime, which would require you to acquire new tokens frequently.
  - Future versions of vSphere might require the use of holder-of-key tokens.

The following figure shows a vCenter client that uses a SAML token to establish a session with a vCenter Server.

Figure 1-1. Single Sign-On in the vSphere Environment – vCenter Server LoginByToken



The vCenter client also operates as a vCenter Single Sign On client. The vCenter Single Sign On client component handles communication with the vCenter Single Sign On Server.

1 The vCenter Single Sign On client sends a token request to the vCenter Single Sign On Server. The request contains information that identifies the principal. The principal has an identity in the identity store. The principal may be a user or it may be a software component. In this scenario, the principal is the user that controls the vCenter client.

2 The vCenter Single Sign On Server uses the identity store to authenticate the principal.

3 The vCenter Single Sign On Server sends a response to the token request. If authentication is successful, the response includes a SAML token.

4 The vCenter client connects to the vCenter Server and calls the `SessionManager` method `LoginByToken` method. The login request contains the SAML token.

The figure shows the vCenter Server, vCenter Single Sign On Server, and identity store as components running on separate machines. You can use different vCenter Single Sign On configurations.

- A vCenter Single Sign On Server can operate as an independent component running on its own machine. The vCenter Single Sign On Server can use a remote identity store or it can manage user accounts in its own internal identity store.

- A vCenter Single Sign On Server can operate as an embedded component running on the vCenter Server machine. In this configuration, the vCenter Single Sign On Server can use a remote identity store, its own internal identity store, or it can access user accounts on the vCenter Server machine.

For information about installing and configuring the vCenter Single Sign On Server, see *vSphere Installation and Setup* and *vSphere Security* in the VMware Documentation Center.

vCenter Single Sign On Client API

The vCenter Single Sign On client API is described in the WSDL (Web Service Definition Language) file that is included in the vCenter Single Sign On SDK. This API defines a set of request operations that correspond to the WS-Trust 1.4 bindings. The set of operations includes `Issue`, `Renew`, `Validate`, and `Challenge` requests.

- `Issue` – Obtains a token from a vCenter Single Sign On Server.
- `Renew` – Renews an existing token.
- `Validate` – Validates an existing token.
- `Challenge` – Part of a negotiation with a vCenter Single Sign On Server to obtain a token.

The vCenter Single Sign On SDK includes Java and C# bindings for the vCenter Single Sign On WSDL. The SDK also contains sample code that demonstrates client-side support for the WS-SecurityPolicy standard. Security policies specify the elements that provide SOAP message security. To secure SOAP messages, a client inserts digital signatures, certificates, and SAML tokens into the SOAP headers for vCenter Single Sign On requests.

- The Java sample includes a JAX-WS implementation of SOAP header methods that support the vCenter Single Sign On security policies.
- The C# sample uses the .NET services for SOAP header manipulation.

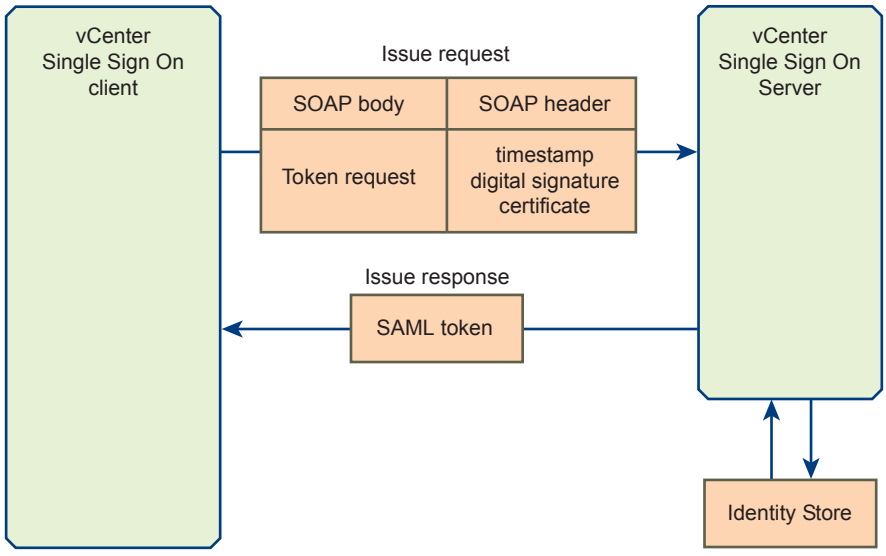
See "vCenter Single Sign On Security Policies" on page 8 and "vCenter Single Sign On SDK" on page 11.

Acquiring a SAML Token

To obtain a security token from a vCenter Single Sign On Server, the vCenter Single Sign On client calls the `Issue` method, which sends a SOAP message that contains a token request and authentication data. This section describes a token request that uses a certificate to obtain a holder-of-key token. When the client creates the token request, it also inserts timestamp, signature, and certificate data into the SOAP security header.

The following figure represents the content of an `Issue` request and the response containing a SAML token.

Figure 1-2. Issue - vCenter Single Sign On Token Request and Response



The vCenter Single Sign On SDK provides Java packages that support SOAP header manipulation. Clients that use C# bindings use .NET services to perform SOAP header manipulation.

When the vCenter Single Sign On Server receives the issue request, it performs the following operations to generate a token:

- Uses the timestamp to validate the request.
- Validates the certificate.
- Uses the certificate to validate the digital signature.
- Uses the certificate subject to authenticate the request. Authentication is obtained from the identity store that is registered with the vCenter Single Sign On Server.
- Generates a token that specifies the principal – the vCenter Single Sign On client – as the token subject.

vCenter Single Sign On Security Policies

Web service security policies define the requirements for secure communication between a Web service and a client. vCenter Single Sign On security policies are based on the WS-Policy framework and WS-SecurityPolicy specifications. A policy identifies specific elements for token requests. Based on the policy requirements, a vCenter Single Sign On client will insert data into the SOAP security header for the token request.

vCenter Single Sign On defines security policies for end user access, solution access, and for token exchange. The policies stipulate the following elements:

- Security certificates (x509V3, x509PKIPathV1, x509PKCS7, or WssSamlV20Token11)
- Message timestamps
- Security binding (transport)
- Encryption algorithm (Basic256Sha256)

vCenter Single Sign On security policies specify that the body of the SOAP message for a holder-of-key token must be signed. Bearer tokens require only the username and timestamp tokens.

NOTE The vCenter Single Sign On Server issues SAML tokens to represent client authentication. The standards documentation also uses the term “token” to refer to claims and certificate data that is inserted into SOAP security headers.

The following table shows the vCenter Single Sign On policies and identifies the requirements for each policy. The vCenter Single Sign On WSDL defines these policies for use with the vCenter Single Sign On methods.

Table 1-1. vCenter Single Sign On Policies

Policy	Description
STSSecPolicy	Defines the transport policy and algorithm suite for all communication with the vCenter Single Sign On Server: ■ Certificate-based server-side SSL authentication. ■ HTTPS transport binding using NIST (National Institute of Standards and Technology) Basic256Sha256 encryption algorithm. The HTTPS token is used to generate the message signature. ■ Request security header must contain a timestamp.
IssueRequestPolicy	Defines the security policy for Issue token requests. IssueRequestPolicy specifies either username token (signed), username token (plaintext password), X509 certificate, or holder-of-key token authentication. You specify username/password or X509 certificate credentials to obtain a vCenter Single Sign On token. If you obtain a holder-of-key token, you can use that token for subsequent Issue requests. Username token (signed) authentication: ■ X509 endorsing supporting token (WssX509V3Token11, WssX509PkiPathV1Token11, or WssX509Pkcs7Token10) ■ WssUsernameToken11 signed supporting token Username token (plaintext password) authentication: ■ WssUsernameToken11 signed supporting token X509 certificate authentication: ■ X509 endorsing supporting token (WssX509V3Token11, WssX509PkiPathV1Token11, or WssX509Pkcs7Token10) Holder-of-Key token authentication: ■ WssSamlV20Token11 assertion referenced by a KeyIdentifier ■ Token must be used to sign the SOAP message body.
RenewRequestPolicy	Defines the security policy for Renew token requests. The request must contain one of the following endorsing supporting tokens. The SOAP message body must be included in the signature generated with the token. ■ WssX509V3Token11 ■ WssX509PkiPathV1Token11 ■ WssX509Pkcs7Token10

vCenter Single Sign On SDK Support for vCenter Single Sign On Security Policies

Security policy support is determined by the programming language that you use to write your client.

C# Clients

Your vCenter Single Sign On client can use the .NET services that support web service security policies. See "Security Policies" on page 29.

Java Clients

The vCenter Single Sign On SDK provides Java utilities that support the vCenter Single Sign On security policies. Your vCenter Single Sign On client can use these utilities to create digital signatures and supporting tokens, and insert them into SOAP headers as required by the policies. The SOAP header utilities are defined in files that are located in the samples directory:

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

See "Using Handler Methods for SOAP Headers" on page 34.

Connecting to a vCenter Single Sign On Server

When a vCenter Single Sign On client connects to a vCenter Single Sign On server, it must specify the server URL as the endpoint for the token request message. The endpoint specification uses the following format:

```
https://hostname|IPaddress:7444/ims/STSService
```

The port number and path suffix (`7444/ims/STSService`) is required. 7444 is the default port number. See "Sending a Request for a Security Token" on page 36 for an example of setting the endpoint for a token request. You can change the port number during vCenter Single Sign On Server installation.

Token Delegation

Holder-of-key tokens can be delegated to services in the vSphere environment. A service that uses a delegated token performs the service on behalf of the principle that provided the token. A token request specifies a `DelegateTo` identity. The `DelegateTo` value can either be a solution token or a reference to a solution token.

Components in the vSphere environment can use delegated tokens. vSphere clients that use the `LoginByToken` method to connect to a vCenter Server do not use delegated tokens. The vCenter Server will use a vSphere client's token to obtain a delegated token. The vCenter Server will use the delegated token to perform operations on behalf of the user after the user's vCenter session has ended. For example, a user may schedule operations to occur over an extended period of time. The vCenter Server will use a delegated token to support these operations.

Token Lifetime - Clock Tolerance

A SAML token contains information about the lifetime of a token. A SAML token uses the `NotBefore` and `NotOnOrAfter` attributes of the SAML `Conditions` element to define the token lifetime.

<saml2:Conditions NotBefore="2011-10-04T21:39:17.731Z" NotOnOrAfter="2011-10-04T21:39:47.731Z">

During a token's lifetime, the vCenter Single Sign On server considers any request containing that token to be valid and the server will perform renewal and validation operations on the token. The lifetime of a token is affected by a clock tolerance value that the vCenter Single Sign On server applies to token requests. The clock tolerance value accounts for differences between time values generated by different systems in the vSphere environment. The clock tolerance is 10 minutes.

Challenge (SSPI)

The vCenter Single Sign On Server supports the use of SSPI (Security Support Provider Interface) for client authentication. SSPI authentication requires that both the client and server use security providers to perform authentication. At the beginning of a vCenter Single Sign On Server session, the vCenter Single Sign On client and vCenter Single Sign On Server exchange data. Each participant will use its security provider to authenticate the data it receives. The authentication exchange continues until both security providers authenticate the data.

The vCenter Single Sign On client API provides a `challenge` request for client participation in SSPI authentication. The following sequence describes the challenge protocol.

- vCenter Single Sign On client sends an `issue` request to the vCenter Single Sign On Server. The request contains the client credentials.
- vCenter Single Sign On Server uses its security provider to authenticate the client. The Server returns a RequestSecurityTokenResponseType object in response to the `issue` request. The response contains a challenge.
- vCenter Single Sign On client uses its security provider to authenticate the vCenter Single Sign On Server response. To continue the authentication exchange, the client sends a `challenge` request to the vCenter Single Sign On Server. The request contains the resolution to the Server's challenge and it can also contain a challenge from the vCenter Single Sign On client.
- vCenter Single Sign On Server uses its security provider to authenticate the client's response. If there are still problems, the Server can continue the authentication exchange by returning a response with an embedded challenge. If authentication is successful, the vCenter Single Sign On Server returns a SAML token to complete the original `issue` request.

To exchange challenge data, the vCenter Single Sign On client and vCenter Single Sign On Server use the following elements defined for both RequestSecurityTokenType and RequestSecurityTokenResponseType objects.

- Context attribute
- BinaryExchange element

vCenter Single Sign On SOAP Message Structure

The requirements listed in the following table apply to the SOAP message structure in vCenter Single Sign On message exchange.

Table 1-2. vCenter Single Sign On SOAP Message Structure

Element	Message Requirements
SOAP envelope	All <wst:RequestSecurityToken>, <wst:RequestSecurityTokenResponse>, and <wst:RequestSecurityTokenResponseCollection> elements must be sent as the single direct child of the body of a SOAP 1.1 <S11:Envelope> element.
	Use HTTP POST to send all vCenter Single Sign On SOAP messages over an SSL/TLS protected channel. Set the SOAPAction HTTP header field to the appropriate message binding.
	The <wsse:Security> header in an vCenter Single Sign On request must contain a <wsu:Timestamp> element.
SOAP message signature	If a signature is applied to a request then it must include: ■ Either the <S11:Body>, or the WS-Trust element as a direct child of the <S11:Body> ■ The <wsu:Timestamp>, if present, in the <S11:Header>.
	Exclusive canonicalization without comments (xml-exc-c14n) must be used prior to signature generation.
	The signature certificate must either be carried either within a <wsse:BinarySecurityToken> or a <saml:Assertion> within <wsse:Security> header of the <S11:Header>.
	The signature must contain a <wsse:SecurityTokenReference> that uses an internal direct reference to the <wsse:BinarySecurityToken>.

vCenter Single Sign On SDK

The vCenter Single Sign On SDK is distributed as part of the VMware vSphere Management SDK. When you extract the contents of the distribution kit, the vCenter Single Sign On SDK is located in the `ssoclient` sub-directory:

```
VMware-vSphere-SDK-build-num
    eam
    sms-sdk
    ssoclient
        docs
        dotnet
            cs
                samples
        java
            JAXWS
                lib
                samples
        wsdl
    vsphere-ws
```

The following table shows the locations of the contents of the vCenter Single Sign On SDK.

Table 1-3. venter Single Sign On SDK Contents

vCenter Single Sign On SDK Component	Location
C# vCenter Single Sign On samples	**ssoclient/dotnet/cs/samples**
JAX-WS vCenter Single Sign On client binding	ssoclient/java/JAXWS/lib
Java samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/samples
VMware SOAP header utilities	ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers

Table 1-3. venter Single Sign On SDK Contents (Continued)

vCenter Single Sign On SDK Component	Location
General utilities for samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/utils
WS-Security utilities for samples	ssoclient/java/JAXWS/samples/com/vmware/sso/client/wssecurity
vCenter LoginByToken sample	ssoclient/java/JAXWS/samples/com/vmware/vsphere/samples
VMware SOAP header utilties for LoginByToken example	ssoclient/java/JAXWS/samples/com/vmware/vsphere/soaphandlers
Documentation for example code	ssoclient/docs/java/JAXWS/samples/javadoc/index.html
WSDL files	ssoclient/wsdl

vCenter Single Sign On SDK Examples

The vCenter Single Sign On SDK contains both C# and Java examples that show how to acquire, validate, and renew tokens. This manual describes examples that show how to obtain a holder-of-key token from a vCenter Single Sign On Server and how to use that token to login to a vCenter Server.

vCenter Single Sign On Examples - C#

Each example is implemented as a Visual Studio project that is contained in its own subdirectory in the samples directory. Each project subdirectory contains an example implementation and the corresponding Visual Studio project files. The following table lists the example projects.

Table 1-4. VMware SSO Client SDK Sample Files – C#

Location	Visual Studio Project	Description
`SDK/ssoclient/dotnet/cs/samples/`		
	AcquireBearerTokenByUserCredentialSample	Demonstrates how to use username and password credentials to obtain a bearer token.
	AcquireHoKTokenByHoKTokenSample	Demonstrates how to use an existing holder-of-key token to obtain a new holder-of-key token.
	AcquireHoKTokenBySolutionCertificateSample	Demonstrates how to use a solution certificate to obtain a holder-of-key token.
	AcquireHoKTokenByUserCredentialSample	Demonstrates how to use username and password credentials to obtain a holder-of-key token.

vCenter Single Sign On Examples - Java

This manual describes two of the Java examples provided by the VMware SSO Client SDK:

- "vCenter Single Sign On Client Example (JAX-WS)" on page 33. This example shows how to obtain a holder-of-key token from the vCenter Single Sign On Server.
- "LoginByToken Example (JAX-WS)" on page 39. This example shows how to use the token to login to vCenter Server.

The following table lists the sample files in the SDK:

Table 1-5. VMware SSO Client SDK Sample Files – Java

Location	Examples	Description
`SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/samples/`		

Table 1-5. VMware SSO Client SDK Sample Files – Java

Location	Examples	Description
	AcquireBearerTokenByUserCredentialSample.java	Demonstrates how to use username and password credentials to obtain a bearer token.
	AcquireHoKTokenByHoKTokenSample.java	Demonstrates how to exchange one holder-of-key token for another.
	AcquireHoKTokenBySolutionCertificateSample.java	Demonstrates how a solution uses its private key and certificate to acquire a holder-of-key token.
	AcquireHoKTokenByUserCredentialSample.java	Demonstrates how to use username, password, and certificate credentials to obtain a holder-of-key token. See "vCenter Single Sign On Client Example (JAX-WS)" on page 33.
	RenewTokenSample.java	Demonstrates how to renew a holder-of-key token.
	ValidateTokenSample.java	Demonstrates how to validate a token.
`SDK/ssoclient/java/JAXWS/samples/com/vmware/sso/client/soaphandlers/`		
	HeaderHandlerResolver.java	Provides methods to manage the set of header handlers.
	SamlTokenExtractionHandler.java	Extracts a SAML token from the vCenter Single Sign On Server response.
	SamlTokenHandler.java	Adds a SAML token to a SOAP security header.
	SSOHeaderhandler.java	Base class for header handler classes.
	TimeStampHandler.java	Adds a timestamp element to a SOAP security header.
	UserCredentialHandler.java	Adds a username token to a SOAP security header.
	WsSecuritySignatureAssertionHandler.java	Uses SAML token assertion ID, private key, and certificate to sign a SOAP message. For use when using an existing token to acquire a new token.
	WsSecurityUserCertificateSignatureHandler.java	Uses a private key and certificate to sign a SOAP message.
`SDK/ssoclient/java/JAXWS/samples/com/vmware/vsphere/samples/`		
	LoginByTokenSample.java	Demonstrates how to use a SAML token to login to a vCenter Server. See "LoginByToken Example (JAX-WS)" on page 39.
`SDK/ssoclient/java/JAXWS/samples/com/vmware/vsphere/soaphandlers/`		
	HeaderCookieExtractionHandler.java	Extracts the vCenter HTTP session cookie from the response to a connection request.
	HeaderCookieHandler.java	Inserts an HTTP cookie into a request.

vCenter Single Sign On API Reference

2

This chapter contains descriptions of the methods and data structures defined for the vCenter Single Sign On client API.

vCenter Single Sign On Client API Methods

The vCenter Single Sign On client API consists of the following methods:

- Issue
- Renew
- Validate
- Challenge

Issue

Sends a security token request to a vCenter Single Sign On Server.

Method Signature

```
Issue (requestSecurityToken : RequestSecurityTokenType)
               returns RequestSecurityTokenResponseCollectionType
```

Parameter

`requestSecurityToken` : RequestSecurityTokenType – The following `RequestSecurityTokenType` elements are required for an Issue request; the remaining elements are optional.

- `RequestType` – Must be the URL "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue".
- `Sig` attribute (`UseKey` element) – Specifies a security token that contains an existing certificate key for subject confirmation.
- `Context` attribute – Required if you include a BinaryExchangeType element for SSPI authentication.

Return Value

RequestSecurityTokenResponseCollectionType – Set of RequestSecurityTokenResponseType. A response contains a SAML token or a challenge requiring additional authentication data.

Comments

Sends a token request to a vCenter Single Sign On Server. The request message must contain security artifacts as determined by the vCenter Single Sign On policy used for the request. The vCenter Single Sign On Server will authenticate the user credentials in the request. For information about configuring user directory support for authentication, see *vSphere Installation and Setup* and *vSphere Security* in the VMware Documentation Center. If the vCenter Single Sign On Server requires information during SSPI authentication, it will negotiate with the vCenter Single Sign On client by embedding a challenge in the response.

Renew

Renews an existing SAML token.

Method Signature

```
Renew (token : RequestSecurityTokenType) returns RequestSecurityTokenResponseType
```

Parameter

token : RequestSecurityTokenType – Security token request containing a SAML token previously obtained from a vCenter Single Sign On Server. The token must be valid (not expired). The following `RequestSecurityTokenType` elements are required for a `Renew` request; the remaining elements are optional.

- `RequestType` – Must be the URL "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew".
- `RenewTarget` – Identifies the SAML token to be renewed.
- `Sig` attribute (`UseKey` element) – Specifies a security token that contains an existing certificate key for subject confirmation.
- `Context` attribute – Required if you include a BinaryExchangeType element for SSPI authentication.

Return Value

RequestSecurityTokenResponseType – Response containing the renewed token.

Comments

You can renew holder-of-key tokens only. In addition to the the required token request elements shown above, the `Renew` request SOAP header must contain security elements according to the security policy.

Validate

Validates an existing SAML token.

Method Signature

```
Validate (token : RequestSecurityToken) returns RequestSecurityTokenResponseType
```

Parameter

token : RequestSecurityTokenType – Security token request containing a SAML token previously obtained from a vCenter Single Sign On Server. The following `RequestSecurityTokenType` elements are required for a `Validate` request; the remaining elements are optional.

- `RequestType` – Must specify the URL "http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate".
- `ValidateTarget` – Identifies the SAML token to be validated.
- `Sig` attribute (`UseKey` element) – Specifies a security token that contains an existing certificate key.
- `Context` attribute – Required if you include a BinaryExchangeType element for SSPI authentication.

Return Value

RequestSecurityTokenResponseType – Response containing the validated token.

Comments

Performs validation of the token and its subject. It includes but is not limited to validations of the following elements:

- Token signature
- Token lifetime
- Token subject
- Token delegates
- Group(s) to which the subject belongs

Challenge

Extends a token request to verify elements in the request.

Method Signature

```
Challenge (response : RequestSecurityTokenResponseType) returns RequestSecurityTokenResponseType
```

Parameter

response : RequestSecurityTokenResponseType – Contains SSPI data in the `BinaryExchange` element.

Return Value

RequestSecurityTokenResponseType – Response containing the validated token.

Comments

Part of a negotiation with a vCenter Single Sign On Server to resolve issues related to SSPI authentication.

vCenter Single Sign On API Data Structures

Use the following objects for the vCenter Single Sign On methods.

RequestSecurityTokenType

RequestSecurityTokenResponseCollectionType

RequestSecurityTokenResponseType

LifetimeType

RenewingType

KeyTypeOpenEnum

UseKeyType

ParticipantsType

ParticipantType

EndpointReference

BinaryExchangeType

AdviceType

AttributeType

RequestSecurityTokenType

Defines a set of token characteristics requested by the vCenter Single Sign On client. The vCenter Single Sign On client specifies this data object in a call to the Issue, Renew, and Validate methods. The vCenter Single Sign On Server may satisfy a request for a particular characteristic or it may use a different value in the issued token. The response to the token request contains the actual token values. See "RequestSecurityTokenResponseType" on page 19.

The vCenter Single Sign On API supports a subset of the `RequestSecurityTokenType` elements defined in the WS-Trust specification. The following table shows the supported elements and attributes. An item in the table is defined as an element in the WSDL unless explicitly identified as an attribute.

Table 2-1. RequestSecurityTokenType Elements (vCenter Single Sign On)

Element	Datatype	Description
Context	string	`RequestSecurityToken` attribute specifying a URI (Uniform Resource Identifier) that identifies the original request. If you include this in a request, the vCenter Single Sign On Server will include the context identifier in the response. This attribute is required when the request includes a `BinaryExchange` property.
TokenType	string	Identifies the requested token type, specified as a URI (Uniform Resource Identifier). The following list shows the valid token types: ■ urn:oasis:names:tc:SAML:2.0:assertion – for issue and renew requests. ■ http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status – for validation requests.
RequestType	string	Identifies the request type, specified as a URI. The `RequestType` property is required. The following list shows the valid request types: ■ http://docs.oasis-open.org/ws-sx/ws-trust/200512/Issue ■ http://docs.oasis-open.org/ws-sx/ws-trust/200512/Renew ■ http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate
Lifetime	LifetimeType	Time period during which a token is valid. The vCenter Single Sign On Server can ignore the requested lifetime and assign a different lifetime to the token. The lifetime specifies creation and expiration values. This property is optional – used with Issue and Renew requests.
ValidateTarget		Specifies the token to be validated. This property can contain either a reference to the token or it can contain the token itself. The property is required for and used only with the Validate method.
RenewTarget		Specifies the token to be renewed. This property can contain either a reference to the token or it can contain the token itself. This property is required for and used only with the Renew method.
Renewing	RenewingType	Specifies a request for a renewable token. This property is optional. If you do not specify the `Renewing` property, the vCenter Single Sign On Server will issue a renewable token. This property is optional.
DelegateTo		Specifies a security token or token reference for an identity to which the requested token will be delegated. The `DelegateTo` value must identify a solution.
Delegatable	xs:boolean	Indicates whether the requested token can be delegated to an identity. Use this property together with the `DelegateTo` property. The default value for the `Delegatable` property is false.
UseKey	UseKeyType	References a token for subject confirmation. Required for Issue, Renew, and Validate methods.
KeyType	string	String value corresponding to a KeyTypeOpenEnum value. The value is a URI (Uniform Resource Identifier) that specifies the requested key cryptography type. This property is optional.
SignatureAlgorithm	string	Specifies a URI (Uniform Resource Identifier) for an algorithm that produces a digital signature for the token. The following list shows the valid values: ■ `http://www.w3.org/2000/09/xmldsig#rsa-sha1` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha384` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha512`
BinaryExchange	BinaryExchangeType	Contains data for challenge negotation between the vCenter Single Sign On client and vCenter Single Sign On Server.

Table 2-1. RequestSecurityTokenType Elements (vCenter Single Sign On)

Element	Datatype	Description
Participants	ParticipantsType	Specifies the identities of participants that are authorized to use the token.
AdviceSet	AdviceSetType	List of AdviceType.

RequestSecurityTokenResponseCollectionType

Returned by the Issue method. This type contains a response to the request or the requested token. .

Table 2-2. RequestSecurityTokenResponseCollectionType

Element	Datatype	Description
RequestSecurityTokenResponse	RequestSecurityTokenResponseType[ ]	List of token request response objects. The current architecture supports a single token response only

RequestSecurityTokenResponseType

Describes a single token.

Table 2-3. RequestSecurityTokenResponseType Properties (vCenter Single Sign On)

Element	Datatype	Description
Context	string	
		`RequestSecurityTokenResponse` attribute specifying a URI (Uniform Resource Identifier) that identifies the original request. This attribute is included in the response if it was specified in the request.
TokenType	string	
		Identifies the type of token in the response. TokenType is specified as a URI (Uniform Resource Identifier), one of the following: ■ urn:oasis:names:tc:SAML:2.0:assertion – for issue and renew operations. ■ http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status – for validation operations.
Lifetime	LifetimeType	
		Time period during which a token is valid. The lifetime in the token response is the actual lifetime assigned by the vCenter Single Sign On Server. The lifetime specifies creation and expiration values.
RequestedSecurityToken	RequestedSecurityTokenType	
		SAML token.
Renewing	RenewingType	
		Indicates whether or not the token can be renewed. By default, the vCenter Single Sign On Server will issue a renewable token.
BinaryExchange	BinaryExchangeType	
		Contains data for challenge negotiation between vCenter Single Sign On client and vCenter Single Sign On Server.
KeyType	string	
		Indicates whether or not key cryptography is used. The KeyType is a string value corresponding to an enumerated type value. See KeyTypeOpenEnum. The value is a URI (Uniform Resource Identifier) that specifies the key type.

Table 2-3. RequestSecurityTokenResponseType Properties (vCenter Single Sign On) (Continued)

Element	Datatype	Description
SignatureAlgorithm	string	
	Indicates a URI (Uniform Resource Identifier) for an algorithm that produces a digital signature for the token. The following list shows the valid values: ■ `http://www.w3.org/2000/09/xmldsig#rsa-sha1` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha384` ■ `http://www.w3.org/2001/04/xmldsig-more#rsa-sha512`	
Delegatable	xs:boolean	
	Indicates whether the requested token can be delegated to an identity.	
Status	StatusType	
	Indicates the status of the request. The property specifies `Code` and `Reason` values.	

LifetimeType

Specifies the token lifetime. Used in RequestSecurityTokenType and RequestSecurityTokenResponseType.

Table 2-4. LifetimeType Properties

Property	Datatype	Description
created	wsu:AttributedDateTime	Creation time of the token. XML date and time, expressed as a standard time value (Gregorian calendar).
expires	wsu:AttributedDateTime	Time interval during which the token is valid, starting at the `created` time. The time interval is an absolute value specified in seconds.

RenewingType

Specifies token renewal.

Table 2-5. RenewingType Properties

Property	DataType	Description
Allow	xsd:boolean	Specifies a request for a token for which the lifetime can be extended. This property is optional. The default value is `true`.
OK	xsd:boolean	Indicates that the vCenter Single Sign On client will accept a token that can be renewed after it has expired. This property is optional. The default value is `false`. If you specify this property, you must specify a value of `false`. A token that can be renewed after expiration does not provide adequate security.

KeyTypeOpenEnum

Specifies a set of enumerated type values that identify the supported types of key cryptography used for security tokens. The values are URIs (Universal Resource Identifiers).

Table 2-6. KeyType Properties

Enumerated type value	Description
http://docs.oasis-open.org/ws-sx/ws-trust/200512/PublicKey	Specifies asymmetric key cryptography using a combination of public and private keys. Use this key type for holder-of-key tokens.
http://docs.oasis-open.org/wssx/wstrust/200512/Bearer	Indicates a bearer token, which does not require a key to authenticate the token.

UseKeyType

Specifies the URI for an existing key.

Table 2-7. UseKeyType Properties

Property	Datatype	Description
Sig	string	URI (Universal Resource Identifer) that refers to a security token which contains an existing key. If specified, the vCenter Single Sign On Server will use the associated certificate for subject confirmation.

ParticipantsType

Identifies users and services who are allowed to use the token.

Table 2-8. ParticipantsType Properties

Property	Datatype	Description
Primary	ParticipantType	Primary user of the token.
Participant	ParticipantType	List of participants who are allowed to use the token.

ParticipantType

ParticipantType is an end point reference.

Table 2-9. ParticipantType Property

Property	Datatype	Description
	EndpointReference	Specifies a participant represented as a URI.

EndpointReference

Participant identification. The `ReferenceParameters`, `Metadata`, and `any` elements are not used.

Table 2-10. EndpointReference Property

Property	Datatype	Description
name	tns:AttributedURIType	URI that identifies a participant allowed to use a token.

BinaryExchangeType

Specifies a blob (binary large object) that contains data for negotation between the vCenter Single Sign On client and server.

Table 2-11. BinaryExchangeType Attributes

Attribute	Datatype	Description
ValueType	xsd:anyURI	Identifies the type of negotiation.
EncodingType	xsd:anyURI	Identifies the encoding format of the blob.

AdviceType

Specifies additional informational attributes to be included in the issued token. The vCenter Single Sign On client can ignore this data. Advice data will be copied to delegate tokens. This type is used in RequestSecurityTokenType.

Table 2-12. AdviceType Properties

Element/Attribute	Datatype	Description
Advicesource	string	AdviceType attribute specifying a URI representing the identity that provides the advice Attribute elements. This attribute is required.
Attribute	AttributeType	Advice data.

AttributeType

Attribute providing advice data. Used in AdviceType.

Table 2-13. AttributeType Properties

Element/Attribute	Datatype	Description
Name	string	`AttributeType` attribute specifying a URI that is the unique name of the attribute. This attribute is required.
FriendlyName	string	`AttributeType` attribute specifying a human-readable form of the name. This attribute is optional.
AttributeValue	string	List of values associated with the attribute. The `AttributeValue` structure depends on the following criteria: ■ If the attribute has one or more values, the `AttributeType` contains one `AttributeValue` for each value. Empty attribute values are represented by empty `AttributeValue` elements. ■ If the attribute does not have a value, the `AttributeType` does not contain an `AttributeValue`.

vCenter Single Sign On Client Example (.NET)

3

This chapter describes a C# example of acquiring a vCenter Single Sign On 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 a C# example that shows how to use the token to login to a vCenter Server, see "LoginByToken Example (.NET)" on page 27. The code examples in this chapter are based on the following Visual Studio project located in the vCenter Single Sign On SDK .NET `samples` directory:

```
.../SDK/ssoclient/dotnet/cs/samples/AcquireHoKTokenByUserCredentialSample
```

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 a SOAP filter to modify the SOAP security header for the request message. The SOAP filter implementation overrides the Microsoft WSE (Web Services Enhancement) method `CreateClientOutputFilter`.
- The program 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 SOAP filter embeds these elements in the message.

Sample Code

The code examples in the following sections show how to obtain a holder-of-key security token. The code examples are based on the `AcquireHokTokenByUserCredentialSample` project contained in the vCenter Single Sign On SDK. The project is located in the `dotnet` samples directory (`SDK/ssoclient/dotnet/cs/samples/AcquireHokTokenByUserCredentialSample`).

- Project file – `AcquireHokTokenByUserCredentialSample.csproj`
- Sample code – `AcquireHokTokenByUserCredential.cs`
- Declarations that override the .NET `SecurityPolicyAssertion` – `CustomSecurityAssertion.cs`
- SOAP header manipulation code – `CustomSecurityClientOutputFilter.cs`

Sending a Request for a Security Token

To send a request for a security token, the sample specifies username and password assertions to satisfy the security policy, creates a request token, and calls the `Issue` method. The following sequence shows these operations.

1 Create the `STSService` client-side object. This object provides access to vCenter Single Sign On request objects and methods.

2 Specify the URL of the vCenter Single Sign On Server.

3 Create a `SoapContext` object for the security headers.

4 Specify username and password assertions to satisfy the security policy.

5 Provide a remote certificate validation callback. The sample version of this callback does not validate the certificate; it just returns a `true` value.

IMPORTANT This is suitable for a development environment, but you should implement certificate validation for a production environment.

6 Create a token request (`RequestSecurityTokenType`) and 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.

7 Call the `Issue` method. The SSO Server returns a response structure that contains the token.

The following example shows C# code that performs these operations.

Example 3-1. Acquiring a vCenter Single Sign On Token – Sending the Request

```
public static XmlElement GetToken(String[] args)
{
    // 1. Create an SSO Server client-side object
    service = new STSService();

    // 2. Set the SSO Server URL
    service.Url = args[0];

    // 3. SOAP Request Context - Required to add secruity headers
    SoapContext requestContext = service.RequestSoapContext;

    // 4. Create a CustomSecurityAssertion object that specifies username and password
    CustomSecurityAssertion objCustomSecurityAssertion = new CustomSecurityAssertion();
    objCustomSecurityAssertion.Username = args[1].Trim();
    objCustomSecurityAssertion.Password = args[2].Trim();

    // Use the assertions to set the policy
    Policy policy = new Policy();
    policy.Assertions.Add(objCustomSecurityAssertion);
    service.SetPolicy(policy);

    // 5. Establish a validation callback for the token certificate
    ServicePointManager.ServerCertificateValidationCallback +=
        new RemoteCertificateValidationCallback(ValidateRemoteCertificate);
```

```
    // 6. Create a token request
    RequestSecurityTokenType tokenType = new RequestSecurityTokenType();

    // Specify the token type, request type, key type, and signature algorithm
    tokenType.TokenType = TokenTypeEnum.urnoasisnamestcSAML20assertion;
    tokenType.RequestType = RequestTypeEnum.httpdocsoasisopenorgwssxwstrust200512Issue;
    tokenType.KeyType = KeyTypeEnum.httpdocsoasisopenorgwssxwstrust200512PublicKey;
    tokenType.SignatureAlgorithm = SignatureAlgorithmEnum.httpwwww3org200104xmldsigmorersasha256;

    // Set the token creation date/time
    LifetimeType lifetime = new LifetimeType();
    AttributedDateTime created = new AttributedDateTime();
    String createdDate =
        XmlConvert.ToString(System.DateTime.Now, XmlDateTimeSerializationMode.Utc);
    created.Value = createdDate;
    lifetime.Created = created;

    // Set the token expiration time
    AttributedDateTime expires = new AttributedDateTime();
    TimeSpan duration = new TimeSpan(1, 10, 10);
    String expireDate =
        XmlConvert.ToString(DateTime.Now.Add(duration), XmlDateTimeSerializationMode.Utc);
    expires.Value = expireDate;
    lifetime.Expires = expires;

    tokenType.Lifetime = lifetime;

    RenewingType renewing = new RenewingType();
    renewing.Allow = true;
    renewing.OK = true;
    tokenType.Renewing = renewing;

    // 7. Call Issue
    try
    {
        RequestSecurityTokenResponseCollectionType responseToken =
               service.Issue(tokenType);
        RequestSecurityTokenResponseType rstResponse =
               responseToken.RequestSecurityTokenResponse;

        return rstResponse.RequestedSecurityToken;
    }
        catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
        throw ex;
    }
}
```

The following code fragment shows the custom output filter for the custom security assertion. The `CustomSecurityClientOutputFilter` class provides three methods:

- `CustomSecurityClientOutputFilter` constructor – Creates a token for the username and password. It also calls the `GetSecurityToken` method and creates a message signature for the security token.
- `SecureMessage` – An override method for the .NET method `SendSecurityFilter.SecureMessage`. The override method adds tokens and the message signature to the .NET Security element.
- `GetSecurityToken` – creates an X509 security token from a PFX certificate file. PFX is a Public-Key Cryptography Standard format that is used to store a private key and the corresponding X509 certificate.

Example 3-2. Custom Output Filter

```
internal class CustomSecurityClientOutputFilter : SendSecurityFilter
{
    UsernameToken userToken = null;
    X509SecurityToken signatureToken = null;
    MessageSignature sig = null;
```

```
    public CustomSecurityClientOutputFilter(CustomSecurityAssertion parentAssertion)
        : base(parentAssertion.ServiceActor, true)
    {
        userToken = new UsernameToken(parentAssertion.Username.Trim(),
               parentAssertion.Password.Trim(), PasswordOption.SendPlainText);
        signatureToken = GetSecurityToken();
        sig = new MessageSignature(signatureToken);
    }

    ///  SecureMessage
    public override void SecureMessage(SoapEnvelope envelope, Security security)
    {
        security.Tokens.Add(userToken);
        security.Tokens.Add(signatureToken);
        security.Elements.Add(sig);
    }

    /// GetSecurityToken - creates the security token from certificate from pfx file
    internal static X509SecurityToken GetSecurityToken()
    {
        X509Certificate2 certificateToBeAdded = new X509Certificate2();
        string certificateFile = ConfigurationManager.AppSettings["PfxCertificateFile"];
        certificateToBeAdded.Import(certificateFile, "", X509KeyStorageFlags.MachineKeySet);
        return new X509SecurityToken(certificateToBeAdded);
    }
}
```

Solution Certificate Support

Solutions that are integrated into the vSphere environment must perform authentication with the vCenter Single Sign On Server to obtain a SAML token for use in the environment.

The vCenter Single Sign On SDK contains a C# sample that demonstrates how to use a solution certificate to obtain a token (`AcquireHoKTokenBySolutionCertificateSample`). The sample uses a PFX file to obtain the certificate and private key. When you run the sample, you specify the PFX file location and the private key password on the command line:

```
AcquireHoKTokenBySolutionCertificateSample sso-server-url path-to-pfx-file private-key-password
```

- The PFX file is located in the following directory on a vCenter Server:

  ```
  /etc/vmware-vpx/ssl/rui.pfx
  ```

  Copy the `rui.pfx` file from the Server to the system on which you are running the sample.

- The password for the private key is located in the `catalina.properties` file on the vCenter Server:

  ```
  /usr/lib/vmware-vpx/tomcat/conf/catalina.properties
  ```

  The catalina.properties file contains the following definition for the private key password:

  ```
  bio-vmssl.SSL.password=testpassword
  ```

The solution certificate sample uses the `X509Certificate2` constructor to load the certificate. See the sample file `AcquireHoKTokenBySolutionCertificate.cs` in the vCenter Single Sign On SDK.

LoginByToken Example (.NET)

4

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

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. See "vCenter Single Sign On Client Example (.NET)" on page 23 for a description of how to obtain a vCenter Single Sign On token.

To establish a vCenter Server session that is based on SAML token authentication, the client must embed the SAML token in the SOAP header of the `LoginByToken` request. The C# `LoginByToken` example uses the following .NET services to support a single sign on session.

Table 4-1. Microsoft .NET Elements for vCenter Single Sign On Sessions

.NET Element / Namespace	vCenter Single Sign On Usage
`SecurityPolicyAssertion` `Microsoft.Web.Services3.Security`	The sample creates a custom policy assertion derived from the `SecurityPolicyAssertion` class. The custom assertion contains the SAML token and X509 certificate.
`SendSecurityFilter` `Microsoft.Web.Services3.Security`	The sample defines a custom output filter derived from the `SendSecurityFilter` class. The custom filter adds the token and certificate to the outgoing SOAP message.
`ServicePointManager` `System.net`	The sample uses the `ServicePointManager` to specify SSL3 and HTTP 100-Continue behavior.
`ConfigurationManager` `System.Configuration`	The sample uses the `ConfigurationManager` to specify certificate metadata (password and certificate type).
`CookieContainer` `System.Net`	The sample uses the `CookieContainer` class to manage vCenter session cookies.

Persistent vCenter Sessions

A persistent vCenter session relies on a session cookie. When the vCenter Server receives a connection request (`SessionManager.RetrieveServiceContent`), the Server creates a session cookie and returns it in the HTTP header of the response. The client-side .NET framework embeds the cookie in HTTP messages that the client sends to the Server.

The `LoginByToken` request includes the SAML token and client certificate security assertions for client authentication. After successful login, the authentication overhead is no longer needed. The client resets the `VimService` context to eliminate the security overhead. Subsequent client requests will contain the session cookie, which is enough to support the persistent, authenticated session.

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 `LoginByTokenSample` project contained in the vCenter Single Sign On SDK. The project is located in the `dotnet` samples directory (`SDK/ssoclient/dotnet/cs/samples/LoginByToken`).

- Project file – `LoginByToken.csproj`
- Sample code – `LoginByTokenSample.cs`
- SOAP header manipulation code – `CustomSecurityAssertionHok.cs`

Using LoginByToken

The example program uses the following elements and general steps:

- LoginByTokenSample Constructor
- Token Acquisition
- Security Policies
- Connection and Login

LoginByTokenSample Constructor

The `LoginByTokenSample` class constructor creates the following elements to set up access to the vCenter Server.

- `VimService` object – Provides access to vSphere API methods and support for security policies and session cookie managment. It also stores the vCenter Server URL.
- `CookieContainer` – Provides local storage for the vCenter session cookie.
- `ManagedObjectReference` – Manually created `ManagedObjectReference` to retrieve a `ServiceInstance` at the beginning of the session.

The following code fragment shows the `LoginByTokenSample` constructor.

Example 4-1. LoginByTokenSample Constructor

```
// Global variables
private VimService _service;
private ManagedObjectReference _svcRef;
private ServiceContent _sic;
private string _serverUrl;


public LoginByTokenSample(string serverUrl)
{
    _service = new VimService();
    _service.Url = serverUrl;
    _serverUrl = serverUrl;
    _service.CookieContainer = new System.Net.CookieContainer();
    _svcRef = new ManagedObjectReference();
    _svcRef.type = "ServiceInstance";
    _svcRef.Value = "ServiceInstance";
}
```

Token Acquisition

The client must obtain a SAML token from a vCenter Single Sign On Server. See "vCenter Single Sign On Client Example (.NET)" on page 23.

Security Policies

The `LoginByToken` sample creates a custom policy assertion that is derived from the .NET class `SecurityPolicyAssertion`. The assertion class gives the .NET framework access to the SAML token and the X509 certificate.

The sample performs the following operations to set up the security policy and message handling.

- Sets the `ServicePointManager` properties to specify SSL3 and HTTP 100-Continue response handling. 100-Continue response handling supports more efficient communication between the client and vCenter Server. When the client-side .NET framework sends a request to the Server, it sends the request header and waits for a 100-Continue response from the Server. After it receives that response, it sends the request body to the Server.
- Creates an `X509Certificate2` object, specifies the certificate file, and imports the certificate. The certificate file specification indicates a PKCS #12 format file (Public-Key Cryptography Standards) – `PfxCertificateFile`. The file contains the client's private key and public certificate. The `PfxCertificateFile` setting is defined in the `app.config` file in the `LoginByToken` project. The definition specifies the location of the file.
- Creates a custom security assertion to store the SAML token and the certificate. The token and certificate will be included in the policy data for the `LoginByToken` request.
- Defines a custom output filter that is derived from the .NET class `SendSecurityFilter`.

Custom Security Assertion

The following code fragment shows the `LoginByTokenSample` class method `GetSecurityPolicyAssertionForHokToken`. The method returns a `CustomSecurityAssertionHok` instance which overrides the .NET class `SecurityPolicyAssertion`. The security assertion contains the SAML token and the X509 certificate token. This code is taken from the `LoginByToken` project file `samples/LoginByToken/CustomSecurityAssertionHok.cs`.

Example 4-2. Setting Up Security Policies

```
private SecurityPolicyAssertion GetSecurityPolicyAssertionForHokToken(XmlElement xmlToken)
{

    //When this property is set to true, client requests that use the POST method
    //expect to receive a 100-Continue response from the server to indicate that
    //the client should send the data to be posted. This mechanism allows clients
    //to avoid sending large amounts of data over the network when the server,
    //based on the request headers, intends to reject the request
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Ssl3;

    X509Certificate2 certificateToBeAdded = new X509Certificate2();
    string certificateFile = ConfigurationManager.AppSettings["PfxCertificateFile"];
    string password = ConfigurationManager.AppSettings["PfxCertificateFilePassword"];
    certificateToBeAdded.Import(certificateFile, password ?? string.Empty,
               X509KeyStorageFlags.MachineKeySet);

    var customSecurityAssertion = new CustomSecurityAssertionHok();
    customSecurityAssertion.BinaryToken = xmlToken;
    customSecurityAssertion.TokenType = strSamlV2TokenType;
    customSecurityAssertion.SecurityToken = new X509SecurityToken(certificateToBeAdded);

    return customSecurityAssertion;
}
```

Custom Output Filter

The following code fragment shows the custom output filter for the custom security assertion. The custom filter provides three methods:

- CustomSecurityClientOutputFilterHok class constructor – Creates token and message signature objects for the SOAP message.
- SecureMessage – An override method for the .NET method `SendSecurityFilter.SecureMessage`. The override method adds the SAML token and message signature to the .NET Security element.
- CreateKeyInfoSignatureElement – Creates an XML document that specifies the SAML token type and ID.

Example 4-3. Output Filter for the Custom SecurityPolicyAssertion

```
internal class CustomSecurityClientOutputFilterHok : SendSecurityFilter
    {
        IssuedToken issuedToken = null;
        string samlAssertionId = null;
        MessageSignature messageSignature = null;

        /// Create a custom SOAP request filter.
        /// (Save the token and certificate.)
        public CustomSecurityClientOutputFilterHok(CustomSecurityAssertionHok parentAssertion)
            : base(parentAssertion.ServiceActor, true)
        {
            issuedToken = new IssuedToken(parentAssertion.BinaryToken,
               parentAssertion.TokenType);
            samlAssertionId = parentAssertion.BinaryToken.Attributes.GetNamedItem("ID").Value;
            messageSignature = new MessageSignature(parentAssertion.SecurityToken);
        }

        ///  Secure the SOAP message before its sent to the server.
        public override void SecureMessage(SoapEnvelope envelope, Security security)
        {
            //create KeyInfo XML element
            messageSignature.KeyInfo = new KeyInfo();
            messageSignature.KeyInfo.LoadXml(CreateKeyInfoSignatureElement());

            security.Tokens.Add(issuedToken);
            security.Elements.Add(messageSignature);
        }

        /// Helper method to create a custom key info signature element.
        /// Returns Key info XML element.
        private XmlElement CreateKeyInfoSignatureElement()
        {
            var xmlDocument = new XmlDocument();
            xmlDocument.LoadXml(@"<root><SecurityTokenReference

                xmlns=""http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
                1.0.xsd""
                xmlns:wsse=""http://docs.oasis-open.org/wss/oasis-wss-wssecurity-secext-1.1.xsd""

                wsse:TokenType=""http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#S
                AMLV2.0"">
                 <KeyIdentifier
                xmlns=""http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
                1.0.xsd""

                ValueType=""http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLID
                "">" + samlAssertionId +
                 @"</KeyIdentifier></SecurityTokenReference></root>");
            return xmlDocument.DocumentElement;
        }
    }
```

Connection and Login

The following code fragment performs the following actions:

- Calls the `LoginByTokenSample` class method `GetSecurityPolicyAssertionForHokToken` (see "Security Policies" on page 29) and adds the security policy to the `VimService` object.

  The `VimService` object contains the following data:

  - vCenter Server URL.
  - SAML token (stored in the security policy assertion).
  - X509 certificate (stored in the security policy assertion).
- Calls the `RetrieveServiceContent` method. The method establishes the connection with the vCenter Server and it returns a session cookie. The session cookie is stored in the cookie container in the `VimService` object.
- Calls the `LoginByToken` method. The .NET framework uses the security policy assertion and the session cookie to construct the login request.
- Calls the `LoginByTokenSample` class method `resetService` to create a new `VimService` object.

Example 4-4. Connection and Login

```
// Construct the security policy assertion
SecurityPolicyAssertion securityPolicyAssertion = null;
securityPolicyAssertion = GetSecurityPolicyAssertionForHokToken(xmlToken);

// Setting up the security policy for the request
Policy policySAML = new Policy();
policySAML.Assertions.Add(securityPolicyAssertion);

// Setting policy of the service
_service.SetPolicy(policySAML);

_sic = _service.RetrieveServiceContent(_svcRef);
if (_sic.sessionManager != null)
{
    _service.LoginByToken(_sic.sessionManager, null);
}
resetService();
```

The following code fragment shows the `resetService` method. The method creates a new `VimService` object and a new cookie container. The method also obtains a new instance of the session cookie.

Example 4-5. The resetService method

```
/// Resetting the VimService without the security policies
/// as we need the policy only for the LoginByToken method
/// and not the other method calls. resetService also maintains the
/// authenticated session cookie post LoginByToken.
///
/// This method needs to be called only after successful
/// login
private void resetService()
{
    var _cookie = getCookie();
    _service = new VimService();
    _service.Url = _serverUrl;
    _service.CookieContainer = new CookieContainer();
    if (_cookie != null)
    {
        _service.CookieContainer.Add(_cookie);
    }
}
```

```
        /// Method to save the session cookie
        private Cookie getCookie()
        {
            if (_service != null)
            {
                var container = _service.CookieContainer;
                if (container != null)
                {
                    var _cookies = container.GetCookies(new Uri(_service.Url));
                    if (_cookies.Count > 0)
                    {
                        return _cookies[0];
                    }
                }
            }
            return null;
        }
```

vCenter Single Sign On Client Example (JAX-WS)

5

This chapter describes a Java example of acquiring a vCenter Single Sign On 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 "LoginByToken Example (JAX-WS)" on page 39. 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:

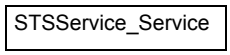
```
import com.vmware.sso.client.soaphandlers.HeaderHandlerResolver;
import com.vmware.sso.client.soaphandlers.SSOHeaderHandler;
import com.vmware.sso.client.soaphandlers.SamlTokenExtractionHandler
import com.vmware.sso.client.soaphandlers.TimeStampHandler;
import com.vmware.sso.client.soaphandlers.UserCredentialHandler;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

This example uses the following handler elements:

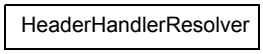
- HeaderHandlerResolver
- SamlTokenExtractionHandler
- TimestampHandler
- UserCredentialHandler
- WsSecurityUserCertificateSignatureHandler (SSOHeaderHandler)

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

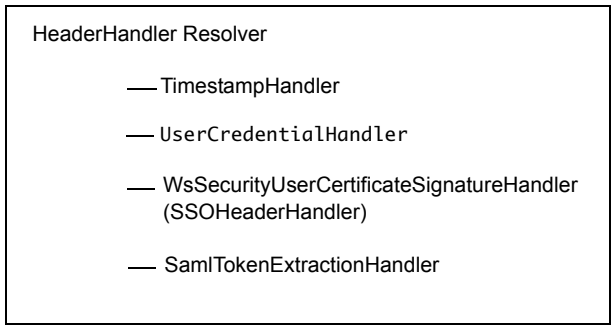
1. Create an STS service object (`STSService_Service`). This object will bind the handlers to the request and provide access to the issue method.



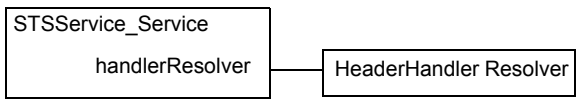
2. Create a handler resolver object (`HeaderHandlerResolver`). This object acts as a receptacle for the handlers.



3. 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.



4. 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" on page 36.

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" on page 36.

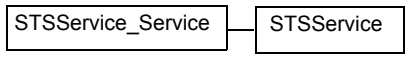
Example 5-1. Acquiring a vCenter Single Sign On Token – Soap Handlers

```
/*
 * 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);
```

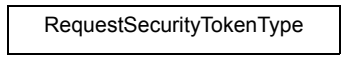
Sending a Request for a Security Token

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.

5 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" on page 34.

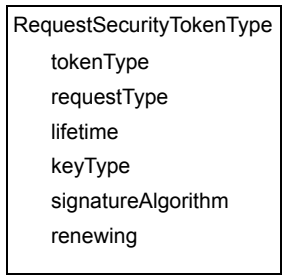


6 Create a token request (`RequestSecurityTokenType`). Your vCenter Single Sign On client will pass the token request to the `Issue` method. The `Issue` method will send the token request in the body of the SOAP message. This example sets the token request fields as appropriate for a holder-of-key token request.

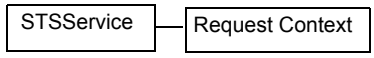


7 Set the token request fields.

- 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



8 Set the endpoint address for the token request.



9 Call the Issue method.

10 Handle the response from the vCenter Single Sign On Server.

The following example shows Java code that performs these operations.

Example 5-2. 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();
```

LoginByToken Example (JAX-WS) 6

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

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. See "vCenter Single Sign On Client Example (JAX-WS)" on page 33 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. 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 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, 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

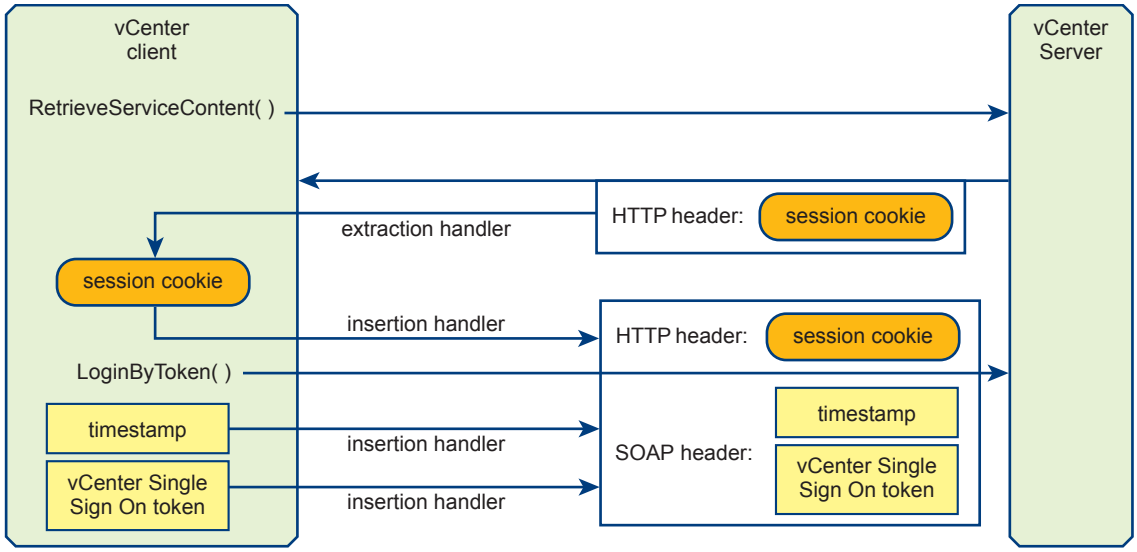
To use a vCenter Single Sign On token to login to a vCenter Server, the example uses header handlers to manipulates the HTTP and SOAP header elements of the login request. After establishing a handler, subsequent requests automatically invoke the handler.

- An extraction handler obtains the HTTP session cookie provided by the vCenter Server. 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.

Figure 6-1. Starting a vCenter Session



IMPORTANT Every call to the vCenter Server 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.

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 "vCenter Single Sign On Client Example (JAX-WS)" on page 33. The SOAP handler files are located in the vCenter Single Sign On client `soaphandlers` directory:

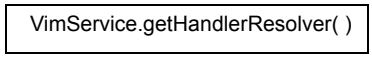
  `samples/com/vmware/sso/client/soaphandlers`

Saving the vCenter Server Session Cookie

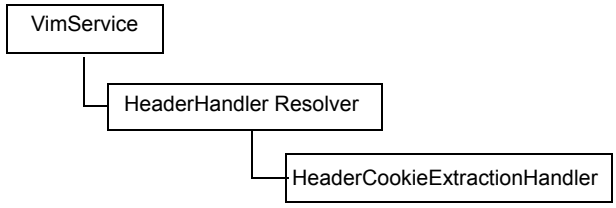
The code fragment in this section establishes an HTTP session with the vCenter Server and saves the HTTP session cookie.

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

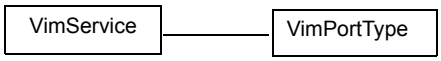
1 Use the `getHandlerResolver` method to save the default message handler. To use the HTTP and SOAP message handlers, you must first save the default message handler so that you can restore it after login. The HTTP and SOAP message handlers impose overhead that is unneccessary after login.



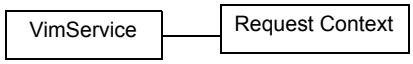
2 Set the cookie handler. The `HeaderCookieExtractionHandler` method retrieves the HTTP cookie.



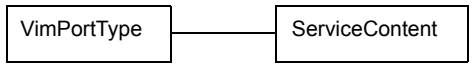
3 Get the VIM port. The VIM port provides access to the vSphere API methods, including the `LoginByToken` method.



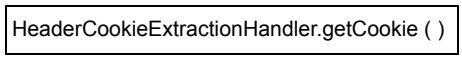
4 Set the request context endpoint address to the vCenter Server URL.



5 Retrieve the ServiceContent. This method establishes the HTTP connection and sets the session cookie.



6 Extract the cookie and save it for later use.



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

Example 6-1. 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.
```

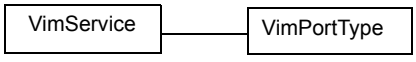
```
 */
ManagedObjectReference SVC_INST_REF = new ManagedObjectReference();
SVC_INST_REF.setType("ServiceInstance");
SVC_INST_REF.setValue("ServiceInstance");

/*
 * 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

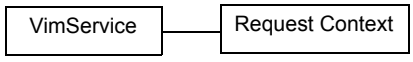
The code fragment in this section sets up the message handlers and calls the `LoginByToken` method. The following sequence describes the steps and shows the corresponding objects and methods.

1 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.

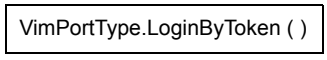
2 Get the VIM port.



3 Set the connection endpoint in the HTTP request context.



4 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.



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

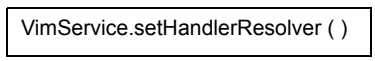
Example 6-2. Using LoginByToken

```
/*
 * 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);
```

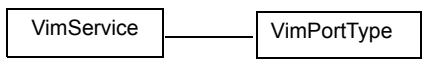
Restoring the vCenter Server Session Cookie

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.

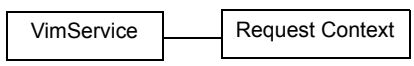
1. Restore the default message handler. The handlers used for `LoginByToken` are not used in subsequent calls to the vSphere API.



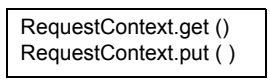
2. Get the VIM port.



3. Set the connection endpoint in the HTTP request context.



4. Set the HTTP request header (vCenter session cookie).



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 "Sample Code" on page 40.

Example 6-3. 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);
```

Index