IBM Security Verify Access
Version 10.0.1
December 2020

Federation Configuration topics

IBM

Contents

Tables

Chapter 1. Federation overview

IBM® Security Verify Access provides a Federation Module so that collaborating organizations can gain secure access to each other's applications. With federated access, you have a secure, seamless sign-on experience to external applications, helping to eliminate the need for providing multiple user IDs and passwords.

By definition, a *federation* is a relationship in which the participating entities agree to use the same technical standard, enabling access to data and resources of one another. It consists of one or more service providers (SP) and an identity provider (IdP). An IdP is a partner in a federation that can authenticate the identity of a user. A service provider is a company or program that provides a business function as a service.

The Federation Module provides the following functions:

- Federated single sign-on (SSO) for users across multiple applications.
- Support for SAML 2.0, WS-Federation, and OpenID Connect protocols for federated access.
- Pre-integrated federation connectors to popular cloud applications.

Activate the Security Verify Access Platform and Federation Module to set up federations.

Chapter 2. Upgrading configuration

After upgrading your Security Verify Access appliance, be aware of some changes to your configuration.

Point of contact profile configuration after an upgrade

The point of contact profile configuration process changed compared to how it was handled before release 9.0.1.

In releases before 9.0.1, the point of contact was configured by using **`poc.*`** advanced configuration parameters. The upgrade process maps these values to callback parameters in a point of contact profile.

Depending on your upgrade scenario, the upgrade makes the following changes:

Table 1. Point of contact profile configuration upgrade scenarios and changes

Upgrade scenario	**Changes**
From a fresh 9.0.1 installation to 9.0.2 or later	None.
.	
From a previously upgraded 9.0.1 to 9.0.2 or later	• Copies the values of the **`poc.*`** advanced configuration parameters to the point of contact profile named **`Advanced configuration`**. The values of the equivalent callback parameters are shown in Table 2 on page 4. • Deletes the **`poc.*`** advanced configuration properties that were moved to the **`Advanced configuration`** profile. The only way that you can update these parameters now is by using the callback parameters. • Makes the **`Advanced configuration`** profile editable. You can now edit the callback parameters for this profile. • Does not change the other point of contact profiles.
From any release before 9.0.1 (for example, 9.0 or 8.0.*) to 9.0.2 or later.	• Keeps the read-only current profile, which was on the appliance before the upgrade. This current profile is called **`Advanced configuration`**. • Removes the other obsolete profiles. • Copies the values of the **`poc.*`** advanced configuration parameters to the point of contact profile named **`Advanced configuration`**. The values of the equivalent callback parameters are shown in Table 2 on page 4. • Deletes the **`poc.*`** advanced configuration properties that were moved to the **`Advanced configuration`** profile. The only way that you can update these parameters now is by using the callback parameters. • Makes the **`Advanced configuration`** profile editable. You can now edit the callback parameters for this profile.

There are also three preconfigured point of contact profiles which support three EAI authentication methods.

For more information, see:

- Creating a point of contact profile

- Callback parameters and values
- "Point of contact advanced configuration property updates " on page 4

Point of contact advanced configuration property updates

The use of several point of contact advanced configuration properties has changed; some properties are deprecated and one is removed after an upgrade.

Table 2 on page 4 shows the status of the point of contact advanced configuration properties:

Table 2. Point of contact advanced configuration properties status and mapping to callback parameters

Callback type	Advanced configuration property names	Status	New callback parameter name
signIn	poc.signIn.userRequestHeader	Deprecated	fim.user.request.header.name
signIn	poc.signIn.attributesResponseHeader	Deprecated	fim.attributes.response.header.name
signIn	poc.signIn.groupsResponseHeader	Deprecated	fim.groups.response.header.name
signIn	poc.signIn.serverResposeHeader	Deprecated	fim.server.response.header.name
signIn	poc.signIn.targetResponseHeader	Deprecated	fim.target.response.header.name
signIn	poc.signIn.userResponseHeader	Deprecated	fim.user.response.header.name
signIn	poc.signIn.userSessionResponseHeader	Deprecated	fim.user.session.id.response.header.name
signIn	poc.signIn.credResponseHeader	Deprecated	fim.cred.response.header.name
signIn	poc.signIn.urlEncodingEnabled	Deprecated	url.encoding.enabled
signIn	poc.signIn.authenticationLevelResponseHeader	Removed from Advanced Configuration panel after an upgrade	None.
signOut	poc.signOut.userSessionRequestHeader	Deprecated	fim.user.session.id.request.header.name
signOut	poc.signOut.userRequestHeader	Deprecated	fim.user.request.header.name
localId	poc.localIdentity.attributesRequestHeader	Deprecated	fim.attributes.request.header.name
localId	poc.localIdentity.credRequestHeader	Deprecated	fim.cred.request.header.name
localId	poc.localIdentity.groupsRequestHeader	Deprecated	fim.groups.request.header.name
localId	poc.localIdentity.userRequestHeader	Deprecated	fim.user.request.header.name
authenticate	poc.websealAuth.userRequestHeader	Deprecated	fim.user.request.header.name
authenticate	poc.websealAuth.authenticationMacros	Deprecated	authentication.macros
authenticate	poc.websealAuth.authLevel	Continue use in Advanced Configuration panel	None.

Table 2. Point of contact advanced configuration properties status and mapping to callback parameters(continued)

Callback type	Advanced configuration property names	Status	New callback parameter name
authenticate	poc.otp.authLevel	Continue use in Advanced Configuration panel	None.
authenticate	poc.otp.backwardCompatibilityEnabled	Continue use in Advanced Configuration panel	None.
authnPolicy	poc.authPolicy.allowRequestOverride	Continue use in Advanced Configuration panel	None.
authnPolicy	poc.authPolicy.authLevel	Continue use in Advanced Configuration panel	None.
authnPolicy	poc.authPolicy.authType	Continue use in Advanced Configuration panel	None.

See more details in "Point of contact profile configuration after an upgrade" on page 3.

Change in OpenID Connect relying party mapping rule

If you're upgrading from version 9.0, you must change the location of the attribute values for issuing authority (iss) and subject (sub). If you don't make these changes, the existing OpenID Connect relying party custom mapping rules fail.

The attribute values for issuing authority (`iss`) and subject (`sub`) are now in the attribute container of the Secure Token Service Universal User (STSUU). In 9.0, these attribute values were in the context attributes.

For example, the following attributes are in the attribute container for versions 9.0.1 or later:

```
stsuu.getAttributeContainer().getAttributeValueByName("iss");
stsuu.getAttributeContainer().getAttributeValueByName("sub");
```

Action: Change your mapping rules to specify the correct location of the attribute values.

Chapter 3. SAML Federations Overview

The Federation Module supports SAML 1.1 and 2.0 federations.

SAML (Security Assertion Markup Language) is a protocol that you can use to perform federated single sign-on from identity providers to service providers. In federated single sign-on, users authenticate at identity provider. Service providers consume the identity information asserted by identity providers.

SAML relies on the use of SOAP, among other technologies, to exchange XML messages over computer networks. The XML messages are exchanged through a series of requests and responses.

In this process, one of the federation partners sends a request message to the other federation partner. Then, that receiving partner immediately sends a response message to the partner who sent the request.

The SAML specifications include descriptors to establish a federation, initialize, and manage single sign-on. The following descriptors specify the structure, content of the messages, and the way the messages are communicated between partners and users.

Assertions
XML-formatted tokens that are used to transfer user identity information, such as the authentication, attribute, and entitlement information, in the messages.

Protocols
The types of request messages and response messages that are used for obtaining authentication data and for managing identities.

Bindings
The communication method that is used to transport the messages.

Profiles
Combinations of protocols, assertions, and bindings that are used together to create a federation and enable federated single sign-on.

You and your partner must use the same SAML specification and agree on which protocols, bindings, and profiles to use.

SAML 1.1

IBM Security Verify Access supports SAML 1.1.

If you and your partner choose to use SAML 1.1 in your federation, you need to understand the SAML 1.1 support that is provided in IBM Security Verify Access.

Assertions

The assertions created by IBM Security Verify Access contain authentication statements, which assert that the principal (that is, the entity requesting access) was authenticated. Assertions can also carry attributes about the user that the identity provider wants to make available to the service provider.

Assertions are usually passed from the identity provider to the service provider.

The following variables control the content of the assertions created by IBM Security Verify Access:

- The specification (SAML 1.1) that you select when you establish a federation.
- The definitions used in the IBM Security Verify Access identity mapping method that you configure.

Identity mapping specifies how identities are mapped between federation partners.

The IBM Security Verify Access identity mapping method can either be a custom mapping module or a JavaScript mapping rule.

Protocol

In IBM Security Verify Access, SAML 1.1 uses a simple request-response protocol to make authentication requests.

Binding

SAML 1.1 uses both plain HTTP (using browser redirects) or SOAP for the transportation of messages. The *profile* used in the federation further specifies how the communication of the messages takes place.

Profiles

SAML 1.1 specifies two options for profiles:

Browser artifact
Browser artifact uses SOAP-based communications (also called the SOAP backchannel) to exchange an artifact during the establishment and use of a trusted session between an identity provider, a service provider, and a client (browser).

Browser POST
Browser POST uses a self-posting form during the establishment and use of the trusted session between an identity provider, a service provider, and a client (browser).

IBM Security Verify Access supports browser artifact by default when you select SAML 1.1 as the profile for your federation. However, you can use browser POST in your federation on a per-partner basis. For example, if you are a service provider, you can specify that your identity provider partner uses Browser POST when you configure that partner. If you are an identity provider, you can enable the IBM PROTOCOL extension when configuring a SAML 1.1 federation.

The URL that is used to initiate single sign-on differs depending on whether the identity provider is using this extension. For more information about URLs, see "SAML 1.1 initial URL" on page 10.

SAML 2.0

The Federation Module relies on the SAML 2.0 specification to establish a federation and to initialize and manage single sign-on.

Assertions

The assertions contain authentication statements. These authentication statements assert that the principal (that is, the entity that requests access) was authenticated. Assertions can also carry attributes about the user that the identity provider wants to make available to the service provider.

Assertions are typically passed from the identity provider to the service provider.

The content of the assertions that are created is controlled by the SAML 2.0 specification. Select these assertions when you establish a federation. You can also select these assertions by the definitions that are used in the identity mapping method that you configure.

The identity mapping method can either be a custom mapping module or a JavaScript mapping rule. The identity mapping also specifies how identities are mapped between federation partners.

Protocols

SAML 2.0 defines several request-response protocols that correspond to the action that is being communicated in the message. The SAML 2.0 protocols that are supported are:

- Authentication request
- Single logout
- Artifact resolution
- Name identifier management

Note: The Enhanced Client or Proxy (ECP) flow is currently not supported by Security Verify Access.

SAML profiles

SAML profiles combine protocols, assertions, and bindings to create a federation and enable federated single sign-on.

The following profiles are supported:

Web browser single sign-on

This profile provides options regarding the initiation of the message flow and the transport of the messages:

Flow initiation

The message flow can be initiated from the identity provider or the service provider.

Bindings

The following bindings can be used in the Web browser SSO profile:

- HTTP redirect
- HTTP POST
- HTTP artifact

The choice of binding depends on the type of messages being sent. For example, an authentication request message can be sent from a service provider to an identity provider using HTTP redirect, HTTP POST, or HTTP artifact. The response message can be sent from an identity provider to a service provider by using either HTTP POST or HTTP artifact. A pair of partners in a federation does not need to use the same binding.

Single Logout

The Single Logout profile is used to terminate all the login sessions currently active for a specified user within the federation. A user who achieves single sign-on to a federation establishes sessions with more than one participant in the federation.

The sessions are managed by a session authority, which in many cases is an identity provider. When the user wants to end sessions with all session participants, the session authority can use the single logout profile to globally terminate all active sessions.

This profile provides options regarding the initiation of the message flow and the transport of the messages:

Flow initiation

The message flow can be initiated from the identity provider or the service provider.

Bindings

The following bindings can be used in the Single Logout profile:

- HTTP redirect
- HTTP POST
- HTTP artifact
- SOAP

Name Identifier Management

The Name Identifier Management profile manages user identities that are exchanged between identity providers and service providers.

This profile can be used by identity providers or service providers to inform their partners when there is a change in user aliases.

This profile can also be used by identity providers or service providers to terminate user linkages at the partners.

To manage the aliases, the Federation module uses a function that is called the *alias service*. The alias service stores and retrieves aliases that are related to a federated identity. User aliases are stored and retrieved from high-volume database.

This profile provides options regarding the initiation of the message flow and the transport of the messages:

Flow initiation
The message flow can be initiated from the identity provider or the service provider.

Bindings
The following bindings can be used in the Web browser SSO profile:

- HTTP redirect
- HTTP POST
- HTTP artifact
- SOAP

SAML 1.1 initial URL

The intersite transfer service URL is where the sign-on request process begins in a SAML 1.1 federation. The URL for initiating a single sign-on request has the following syntax:

Syntax

```
https://identity_provider_hostname:port_number/sps/junction_name
  federation_name/saml11/login?TARGET= target_application_location
  [optional query strings]
```

Elements

identity_provider_hostname
The host name of the reverse proxy server of the identity provider.

port_number
The port number of the reverse proxy server. The default value is 443.

sps
The designation for IBM Security Verify Access server the This element cannot be changed.

junction_name
The name of the junction created on the reverse proxy server. For example, `isva`

federation_name
The name of the SAML 1.1 federation.

saml11
The designation of the SAML protocol you choose to use in your federation.

login
This element indicates what type of endpoint is using the port. **login** is used for the intersite transfer service.

You have the option of using either, both, or neither of the optional query strings (**SP_PROVIDER**) and (**PROTOCOL**), see the following examples:

TARGET
The URL of the target application that a user can log on to using single sign-on.

SP_PROVIDER_ID
The value of query string specifies the provider ID of the service provider that is the target of the single sign-on request. This query string is optional but might be necessary. The use of this query string removes any ambiguity about which service provider is the target of the single sign-on request.

Without this query string, the service provider is determined by matching the *`URI://hostname[:port]`* of the URL in the TARGET query string to the *`URI://hostname[:port]`* of the provider ID for the service provider partner that is configured for the federation. This parameter is used with requests that are initiated at the identity provider.

PROTOCOL

The value of this parameter specifies the type of single sign-on profile (browser artifact or browser POST) that can be used for the single sign-on request. The syntax of the extension is `PROTOCOL=[BA|POST]`, with BA indicating Browser Artifact and POST indicating Browser POST. The query string overrides local identity provider configuration.

The use of the extension is optional. When the extension is not present, the profile choice is determined by the configuration file settings. To use this extension, you must enable the IBM PROTOCOL extension setting during the configuration steps for creating a SAML 1.1 federation on an identity provider.

These query strings can be used individually or in combination. For example, the URL used to initiate single sign-on, when the SP_PROVIDER_ID is used but the PROTOCOL extension is not, has the following syntax:

```
https://intersite_transfer_service_URL?SP_PROVIDER_ID=
  provider_ID_of_service_provider&TARGET=target_application_URL
```

With the SP_PROVIDER_ID and the PROTOCOL extension, the URL has the following syntax:

```
https://intersite_transfer_service_URL?SP_PROVIDER_ID=
  provider_ID_of_service_provider&TARGET=target_application_URL
  &PROTOCOL=[BA|POST]
```

Examples

Single sign-on URL, without the optional parameters:

The following example shows the single sign-on URL for an identity provider using a federation named `ipfed`, the SAML 1.1 protocol, a service provider with a provider ID of `https://sp.example.com:443`, and an application called `snoop`:

```
https://idp.example.com:443/sps/ipfed/saml11/login?TARGET=
  https://sp.example.com:443/snoop/
```

Single sign-on URL, when SP_PROVIDER_ID and PROTOCOL extension *are* used:

The following example shows a URL that is used to initiate single sign-on when the IBM PROTOCOL extension *is* used. In this example, even if the identity provider is configured to use a POST profile for the service provider named `sp`, the following use of the PROTOCOL extension would force the identity provider to use the browser artifact profile:

```
https://idp.example.com:443/isam/sps/ipfed/saml11/login?SP_PROVIDER_ID=
  https://sp.example.com:443/isam/sps/spfed/saml11&TARGET=
  https://sp.example.com:443/isam/
  snoop&PROTOCOL=BA
```

Single sign-on URL, when SP_PROVIDER_ID is used but the PROTOCOL extension is *not* used:

The following example shows a URL that is used to initiate single sign-on when the SP_PROVIDER_ID is used but the IBM PROTOCOL extension is *not* used:

```
https://idp.example.com:443/isam/sps/ipfed/saml11/login?SP_PROVIDER_ID=
  https://sp.example.com:443/isam/sps/spfed/saml11&TARGET=
  https://sp.example.com:443/snoop
```

SAML 2.0 endpoints and URLs

Communications within a federation take place through endpoints on the servers of the identity provider and service provider partners.

In a Security Verify Access environment, endpoints fall into two categories:

- Endpoints that are specified by the federation specification (such as SAML 2.0) and are used for partner-to-partner communication.
- Endpoints that end users can access to initiate a single sign-on activity.

All endpoints can be accessed through URLs. The syntax of the URLs is specific to the purpose of the access and whether the access is by a partner or by an end user.

URLs for partner communication

The URLs that are used for partner-to-partner communication, such as the exchange of requests, in SAML 2.0 federations are referred to collectively as *endpoint URLs*. They can also be individually referred to by the name of the protocol and binding or service that they are related to. Administrators who are responsible for installing, configuring, and maintaining the Security Verify Access environment and the partner-to-partner communication in that environment will see references to these endpoint URLs and might find it helpful to understand their purpose. See "Endpoint URL specifications" on page 12.

URLs for user access

While the SAML specifications define the endpoints for partner-to-partner communication, they provide limited or no guidance about the endpoints or methods that end users must use to initiate single sign-on actions. Security Verify Access supports specific URLs for end-user initiation of single sign-on actions.

In a SAML 2.0 federation, single sign-on actions can be initiated at the identity provider site or the service provider site. URLs that can be used by users to initiate a sign-on action are specific to the a single sign-on action, such as initiate a federated sign on, perform a single logout, or end account linkage. They are also specific to whether the action is being initiated at the identity provider or service provider site. In a Security Verify Access environment, the URLs that can be used for initiating sign-on actions are referred to as *profile initial URLs*. Architects and application developers, who design and implement the interactions of their users with the single sign-on process, need to understand profile initial URLs.

Endpoint URL specifications

You must define several endpoints on your point of contact server so that communications can be exchanged between you and your partner.

These endpoints are defined when you configure your federation in Security Verify Access. The endpoints are accessible through URLs and are used by the partners in the federation.

The following types of endpoint URLs initiate single sign-on:

- Single sign-on service
- Assertion consumer service
- Single logout service endpoint
- Artifact resolution service or SOAP
- Name identifier management service

Single sign-on service endpoint URL (IP)

The endpoint on the identity provider point of contact server that receives authentication requests. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/login
```

Where:

isam_hostname
The host name of the reverse proxy server for the identity provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

Assertion consumer service endpoint (SP)
The endpoint on the service provider point of contact server that receives assertions. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/login
```

Where:

isam_hostname
The host name of the reverse proxy server for the service provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

Single logout service endpoint (IP or SP)
The endpoint on the service provider or identity provider point of contact server that receives logout requests. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/slo
```

Where:

isam_hostname
The host name of the reverse proxy server for the service provider or identity provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

Artifact resolution service or SOAP endpoint (IP or SP)
The endpoint on the service provider or identity provider where artifacts are exchanged for SAML messages. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/soap
```

Where:

isam_hostname
The host name of the reverse proxy server for the service provider or identity provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

Name identifier management service endpoint (IP or SP)
The endpoint on the service provider or identity provider that receives messages related to the name ID management. The unauth ACL must be attached to this endpoint.

The syntax of the URL for HTTP redirect, HTTP POST, and HTTP artifact binding is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/mnids
```

The syntax of the URL for SOAP binding is:

```
https://isam_hostname:port_number/junction_name/sps/federation_name/saml20/soap
```

Where:

isam_hostname
The host name of the reverse proxy server for the service provider or identity provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

SAML 2.0 profile initial URLs

In a federated environment, specially formed URLs can be used for user-initiated single sign-on actions. You can initiate a single sign-on flow from the service provider or identity provider.

The following profile initial URLs are supported in a Security Verify Access environment:

- Assertion consumer service
- Single sign-on service
- Single logout service
- Name identifier management service

Assertion consumer service initial URL (SP)
Initiate the single sign-on flow at the service provider. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps
  /federation_name/saml20/logininitial
  ?RequestBinding=RequestBindingType
  &ResponseBinding=ResponseBindingType
  &NameIdFormat=NameIDFormatType
  &IsPassive=IsPassiveValue
  &IncludeIsPassive=IncludeIsPassiveValue
  &ForceAuthn=ForceAuthnValue
  &IncludeForceAuthn=IncludeForceAuthnValue
  &AllowCreate=AllowCreateValue
  &IncludeAllowCreate=IncludeAllowCreateValue
  &AuthnContextClassRef=ClassRefValues
  &AuthnContextDeclRef=DeclarationRefValues
  &AuthnContextComparison=AuthnContectComparisonValue
  &Target=target_application_location
```

Where:

isam_hostname
The host name of the reverse proxy server for the service provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

RequestBindingType
The binding that is used to send the request. The valid values when initiating single sign-on at the service provider are:

- HTTPPost
- HTTPRedirect
- HTTPArtifact

ResponseBindingType
The binding that is used by the responder to return the response. The valid values when initiating single sign-on at the service provider are:

- HTTPPost
- HTTPArtifact

NameIdFormatType
The name ID format to use for name identifiers. Valid values are:

- Transient (anonymous)
- Persistent
- Email

IsPassiveValue

Specifies if the identity provider must take control of the user agent. A value of true means that the identity provider is not permitted to request the user to provide log in credentials. The default value is false.

IncludeIsPassiveValue
Specifies whether to include the IsPassive attribute in the SAML authentication request. The value of the IsPassive attribute is taken from the IsPassive query string parameter. A value of true includes the attribute. The default value is true.

ForceAuthnValue
Specifies if the identity provider authenticates the user. A value of true means that the user must be authenticated. The default value is false.

IncludeForceAuthnValue
Specifies whether to include the ForceAuthn attribute in the SAML authentication request. The value of the ForceAuthn attribute is taken from the ForceAuthn query string parameter.A value of true includes the attribute. The default value is true.

AllowCreateValue
Specifies if new persistent account linkage is performed on the request. The default value is true. To use this parameter, the NameIdFormat must be set to Persistent.

IncludeAllowCreateValue
Specifies whether to include the AllowCreate attribute in the SAML authentication request. The value of the AllowCreate attribute is taken from the AllowCreate query string parameter. A value of true includes the attribute. The default value is true.

ClassRefValues
Specifies one or more string values which identify authentication context class URI references.

DeclarationRefValues
Specifies one or more string values which identify authentication context declaration URI references.

AuthnContectComparisonValue
Specifies the type of comparison used to determine the requested context classes or declarations. The comparison type must be one of the following variables:

- exact
- minimum

- maximum
- better

The default value is exact.

target_application_location
The URL of the application that a user can log on to using single sign-on.

Example:

Single sign-on URL when initiated at the service provider:

```
https://sp.example.com:433/samlsp/sps/spfed/saml20/logininitial
  ?RequestBinding=HTTPPost
  &ResponseBinding=HTTPPost
  &NameIdFormat=Email
  &IsPassive=true
  &ForceAuthn=false
  &Target=https://sp.example.com:433/samlsp/banking
```

Single sign-on service initial URL (IP)
Initiate the single sign-on flow at the identity provider. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps
  /federation_name/saml20/logininitial
  ?RequestBinding=RequestBindingType
  &PartnerId=target_partner_provider_ID
  &NameIdFormat=NameIDFormatType
  &AllowCreate=AllowCreateValue
  &Target=target_application_location
```

Where:

isam_hostname
The host name of the reverse proxy server for the identity provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

RequestBindingType
The binding that is used to send the request to the service provider. The valid values when initiating single sign-on at the identity provider are:

- HTTPPost
- HTTPArtifact

target_partner_provider_ID
The provider ID of the target partner.

NameIdFormatType
The name ID format to use for name identifiers. Valid values are:

- Transient (anonymous)
- Persistent
- Email

AllowCreateValue
Specifies if new persistent account linkage is performed on the request. The default value is false.

target_application_location
This element is URL-encoded and set as the value of the RelayState parameter in the unsolicited response delivered by the identity provider to the service provider. A service provider interprets this value as the URL of the application that a user can log on to using single sign-on.

Example:

Single sign-on URL when initiated at the identity provider:

```
https://idp.example.com:433/samlip/sps/saml20/saml20/logininitial
  ?RequestBinding=HTTPPost
  &NameIdFormat=persistent
  &AllowCreate=true
  &PartnerId=https://sp.example.com:433/samlsp/sps/saml20/saml20
  &Target=https://sp.example.com:9443/banking
```

Single logout service initial URL (IP or SP)
Initiate the SLO flow at either the identity provider or service provider. The unauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps
  /federation_name/saml20/sloinitial
  ?RequestBinding=RequestBindingType
```

Where:

isam_hostname
The host name of the reverse proxy server for the identity provider or service provider.

port_number
The port number of the reverse proxy server.

junction_name
The name of the junction created on the reverse proxy server.

federation_name
The name you assigned to the federation when you created it.

RequestBindingType
The binding that is used to send the request. The valid values are:

- HTTPPost
- HTTPRedirect
- HTTPArtifact
- HTTPSOAP

Examples:

Single logout URL when initiated at the service provider:

```
https://sp.example.com:433/samlsp/sps/spfed/saml20/sloinitial
  ?RequestBinding=HTTPRedirect
```

Single logout URL when initiated at the identity provider:

```
https://idp.example.com:433/samlip/sps/ipfed/saml20/sloinitial
  ?RequestBinding=HTTPPost
```

Name identifier management service initial URL (IP or SP)
Used by the partner to contact the name identifier management server. The anyauth ACL must be attached to this URL. The syntax of the URL is:

```
https://isam_hostname:port_number/junction_name/sps
  /federation_name/saml20/mnidsinitial
  ?RequestBinding=RequestBindingType
  &PartnerId=target_partner_provider_ID
  &NameIdTerminate=name_ID_terminate_value
```

Where:

isam_hostname
: The host name of the reverse proxy server for the identity provider or service provider.

port_number
: The port number of the reverse proxy server.

junction_name
: The name of the junction created on the reverse proxy server.

federation_name
: The name you assigned to the federation when you created it.

RequestBindingType
: The binding that is used to send the request. The valid values are:

- HTTPPost
- HTTPRedirect
- HTTPArtifact
- HTTPSOAP

target_partner_provider_ID
: The provider ID of the target partner.

name_ID_terminate_value
: A value that indicates if the name ID management flow must terminate the name ID mapping. Valid values are:

- True: Ends the account linkage.
- False: Indicates that the name ID flow updates the name identifiers (aliases). False is the default, if you do not explicitly specify a value.

Examples:

Name ID management initiated by the identity provider:

```
https://idp.example.com:443/samlip/sps/ipfed/saml20/mnidsinitial
  ?RequestBinding=HTTPSOAP
  &PartnerId=https://sp.example.com:443/samlsp/sps/spfed/saml20
  &NameIdTerminate=true
```

Name ID management initiated by the service provider:

```
https://sp.example.com:443/samlsp/sps/spfed/saml20/mnidsinitial
  ?RequestBinding=HTTPArtifact
  &PartnerId=https://idp.example.com:443/samlip/sps/ipfed/saml20
  &NameIdTerminate=true
```

Customizing SAML identity mapping

Use mapping rules to map local identities to SAML tokens and to map SAML tokens to local identities.

You can use an attribute source, such as LDAP, for the identity mapping. See Managing attribute sources.

You can use an HTTP external user mapping to map a local identity to a SAML token and to map SAML token to a local identity.

See Managing JavaScript mapping rules for information about how to create or modify mapping rules.

Mapping a local user identity to a SAML 1.1 token

You can map a local identity to a SAML 1.1 token for an identity provider.

The Security Verify Access server places the local user identity information into an XML document that conforms to the security token service universal user (STSUUSER) schema. The identity provider issues a

SAML 1.1 token to the service provider. It generates the SAML 1.1 token based on the local identity of the user. You can customize how the local identity is converted into a SAML 1.1 token by using a mapping rule.

Security Verify Access first converts the local identity to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a SAML 1.1 token.

Your mapping rule does not operate directly on local identity or SAML 1.1 token. Instead, it operates on the STS Universal User. Any modification that you make to an STS Universal User has an impact on the output SAML 1.1 token.

The mapping rule is responsible for the following tasks:

1. Mapping Principal Attr Name to a Principal Name entry. When the token module generates the token, this Principal name is not directly used. Instead, the value in the **Name** field is sent as input to the alias service. The alias service obtains the alias name, name identifier, for the principal, and places the returned alias in the generated token module.

   The type must be valid for SAML. For example:

   ```
   urn:oasis:names:tc:SAML:1.1:assertion
   ```

2. Setting the authentication method to the `password` mechanism. This action is required by the SAML standard.

Mapping a SAML 1.1 token to a local user identity

You can map a SAML 1.1 token to a local identity for a service provider.

A service provider consumes a SAML 1.1 token that is issued by an identity provider. It generates the local identity of the user based on a SAML 1.1 token. You can customize how a SAML 1.1 token is converted into the local identity of the user by using a mapping rule.

Security Verify Access first converts a SAML 1.1 token to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a local identity of the user.

Your mapping rule does not operate directly on the local identity or SAML 1.1 token. Instead, it operates on the STS Universal User. Any modifications that you make on the STS Universal User impacts the output local identity of the user.

Mapping a local identity to a SAML 2.0 token

You can map a local identity to a SAML 2.0 token for an identity provider.

The Security Verify Access server places the local user identity information into an XML document that conforms to the security token service universal user (STSUUSER) schema. The identity provider issues a SAML 2.0 token to the service provider. It generates the SAML 2.0 token based on the local identity of the user. You can customize how the local identity is converted into a SAML 2.0 token by using a mapping rule.

Security Verify Access first converts the local identity to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a SAML 2.0 token.

Your mapping rule does not operate directly on local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modification that you make to an STS Universal User has an impact on the output SAML 2.0 token.

The mapping rule is responsible for the following tasks:

1. Mapping Principal Attr Name to a Principal Name entry. When the token module generates the token, this Principal name is not directly used. Instead, the value in the **Name** field is sent as input to the

alias service. The alias service obtains the alias name, name identifier, for the principal, and places the returned alias in the generated token module.

The type must be valid for SAML. For example:

```
urn:oasis:names:tc:SAML:2.0:assertion
```

2. Setting the authentication method to the `password` mechanism. This action is required by the SAML standard.
3. Setting the audience of the audience restriction condition to the value of the STSUU element `AudienceRestriction`. If this STSUU element is not present, the audience is set to the Provider ID of the federation partner.
4. Populating the attribute statement of the assertion with the attributes in the AttributeList in the In-STSUU. This information becomes custom information in the token.

   Custom attributes might exist that are required by applications that use information that is to be transmitted between federation partners.
5. Specifying whether the assertion conditions should contain the `<saml:OneTimeUse></saml:OneTimeUse>` element. If so, insert a special context attribute into the STSUU as shown:

```
var oneTimeUseAttr = new Attribute("AssertionIncludeOneTimeUse","urn:oasis:names:tc:SAML:2.0:assertion",
 "true");
stsuu.addContextAttribute(oneTimeUseAttr);
```

6. Setting the `NameID` attribute in the assertion with Transient NameId format. This action is useful when you want to specify a name value to use instead of the default UUID that is generated by the runtime for Transient NameID format.

   To replace the UUID, create a principal name attribute of type `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`, with its value provided by user.

   The examples below show the user-provided value *`UserGeneratedTransientId`* but it could be any other value. The value of the specified STSUU principal name will be set as the `NameID` in the SAML assertion.

   Example mapping rule

   ```
   importPackage(Packages.com.tivoli.am.fim.trustserver.sts.uuser);
   var transientNameId = "UserGeneratedTransientId";
   stsuu.addPrincipalAttribute(new Attribute("name",
      "urn:oasis:names:tc:SAML:2.0:nameid-format:transient", transientNameId));
   ```

   Example STSUU values after mapping rule applied

   ```
    <stsuuser:Attribute name="name" type="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
               <stsuuser:Value>UserGeneratedTransientId</stsuuser:Value>
           </stsuuser:Attribute>
   ```

   Example SAML assertion `NameID` with Transient NameId formats

   ```
   <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
                         NameQualifier="https://ip-wga/isam/sps/saml20ip/saml20"
                         SPNameQualifier="https://sp-wga/isam/sps/saml20sp/saml20"
                         >UserGeneratedTransientId</saml:NameID>
   ```
7. Determine if the partner requires a specific `SPNameQualifier` within `NameID` of assertion for transient identifiers. To change `SPNameQualifer` within `NameID` of assertion, insert a special context attribute into the STSUU with a value agreed with partner as shown in the following example:

   ```
   var SPNameQualifierAttr = new
   Attribute("AssertionChangeSPNameQualifier","urn:oasis:names:tc:SAML:2.0:assertion","http://
   sp/target/app");
   stsuu.addContextAttribute(SPNameQualifierAttr);
   ```

Mapping a SAML 2.0 token to a local identity

You can map a SAML 2.0 token to a local identity for a service provider.

A service provider consumes a SAML 2.0 token that is issued by an identity provider. It generates the local identity of the user based on a SAML 2.0 token. You can customize how a SAML 2.0 token is converted into the local identity of the user by using a mapping rule.

Security Verify Access first converts a SAML 2.0 token to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a local identity of the user.

Your mapping rule does not operate directly on the local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modifications that you make on the STS Universal User impacts the output local identity of the user.

Creating a SAML federation

Create a federation by gathering the necessary configuration information for input into the local management interface on the appliance.

To set up a federation, follow these steps:

1. Create and configure a reverse proxy instance to act as the point of contact for the federation. See Chapter 8, “Configuring a reverse proxy point of contact server,” on page 179.
2. Gather the required data. See “Gathering your federation configuration information” on page 21.
3. Use the local management interface to create your role in the federation. See Creating and modifying federation properties.

Next, you can set up your federation partner. See “Creating a SAML partner” on page 39.

Gathering your federation configuration information

Setting up a federation requires that you first gather the required information according to your role in the federation.

Procedure

- If your role in the federation is a SAML 1.1 service provider, fill out this worksheet: “SAML 1.1 service provider worksheet” on page 21
- If your role in the federation is a SAML 1.1 identity provider, fill out this worksheet: “SAML 1.1 identity provider worksheet” on page 23
- If your role in the federation is a SAML 2.0 service provider, fill out this worksheet: “SAML 2.0 service provider worksheet” on page 25
- If your role in the federation is an SAML 2.0 identity provider, fill out this worksheet: “SAML 2.0 identity provider worksheet” on page 32

SAML 1.1 service provider worksheet

If you assume the role of the service provider in the federation, and use SAML 1.1, record your configuration information in the following tables.

Table 3. General information for service provider in SAML 1.1 federation

General Information	Description	Your value
Federation name	The unique name you give to the federation.	

Table 3. General information for service provider in SAML 1.1 federation (continued)

General Information	Description	Your value
Role	The role you provide in the federation. (In these instructions, you are the service provider.)	Service provider
Company name	The name of the company that is creating this provider.	

Table 4. Federation protocol for service provider in SAML 1.1 federation

Federation Protocol	Description	Your value
Protocol	The SAML protocol you and your partner use in the federation.	SAML 1.1

Table 5. Point of contact server information for service provider in SAML 1.1 federation

Point of contact server	Description	Your value
Point of contact server URL	The URL that provides access to the endpoints on the point of contact server.	

Table 6. Single Sign-On settings for service provider in SAML 1.1 federation

Settings	Description	Your value
Enable one-time assertion use enforcement	This setting is to ensure the SAML assertion is used only once.	• `True` • `False`
Include the following attribute types in the SAML assertions	Provide attribute types in the value text box.	A "*" means include all types. It is selected by default.

Table 7. Signature information for service provider in SAML 1.1 federation

Signatures	Description	Your value
Sign Artifact Resolution Requests	A check box that indicates that you will sign request messages. Default value: No signing. The check box is not selected.	One of the following: • Sign request messages. (Select check box.) • Do not sign request messages. (Clear check box.)
Select Signing Key • Keystore in IBM Security Verify Access key service, where the key is stored • Private key you will use to sign request messages	If you select the check box, you must supply the signing key that you will use to sign the requests. **Note:** Be sure you have created the key and imported it into the appropriate keystore in the IBM Security Verify Access key service prior to this task.	• Keystore name • Certificate Label

Table 8. Identity mapping information for service provider in SAML 1.1 federation

Identity mapping	Description	Your value
Identity mapping options • User JavaScript transformation for identity mapping • Use an external web service for identity mapping	If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account. If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

SAML 1.1 identity provider worksheet

If you assume the role of the identity provider in the federation, and use SAML 1.1, record your configuration information in the following tables.

Table 9. General information for identity provider in SAML 1.1 federation

General Information	Description	Your value
Federation name	The unique name you give to the federation.	
Role	The role you provide in the federation. (In these instructions, you are the identity provider.)	Identity provider
Company name	The name of the company that is creating this provider.	

Table 10. Federation protocol information for identity provider in SAML 1.1 federation

Federation Protocol	Description	Your value
Protocol	The SAML protocol you and your partner use in the federation.	SAML 1.1

Table 11. Point of contact server for identity provider in SAML 1.1 federation

Point of Contact Server	Description	Your value
Point of contact server URL	The URL that provides access to the endpoints on the point of contact server.	

Table 12. Single Sign-On settings for identity provider in SAML 1.1 federation

Settings	Description	Your value
Amount of time before the issue date that an assertion is considered valid	The number of seconds that an assertion is considered valid before its issue date. Default value: 60	
Amount of time the assertion is valid after being issued	The number of seconds that an assertion is considered valid after its issue date. Default value: 60	
Include the following attribute types in the SAML assertions	Provide attribute types in the value text box.	A "*" means include all types. It is selected by default.

Table 13. Signing information for identity provider in SAML 1.1 federation

Signatures	Description	Your value
Signature options: • **SAML messages for Browser POST profile are signed** (required) • **Sign SAML messages for artifact profile** (optional)	• When browser POST is used as the profile, SAML messages must be signed. Therefore, it is pre-selected and cannot be deselected. • You have the option of also signing the SAML messages when browser artifact is used.	One of the following: • Sign browser artifact messages. (Select check box.) • Do not sign browser artifact messages. (Clear check box.)
Select Signing Key • Keystore in IBM Security Verify Access key service, where the key is stored • Private key you will use for signing	Because Browser POST messages must be signed, you are required to supply a signing key. If you select to also sign messages when browser artifact is used, the same signing key is used to sign them. **Note:** Be sure you have created the key and imported it into the appropriate keystore in the IBM Security Verify Access key service prior to this task.	• Keystore name • Certificate label

Table 14. SAML Message Settings information for identity provider in SAML 1.1 federation

SAML Message Settings	Description	Your value
Artifact Resolution Service URL	The URL for your artifact resolution endpoint. (**Note:** The value for this field is filled in automatically using the point of contact server URL you specified earlier.)	
Artifact Cache Lifetime (seconds)	The artifact cache lifetime in seconds. Default value: 30 seconds.	

Table 14. SAML Message Settings information for identity provider in SAML 1.1 federation (continued)

SAML Message Settings	Description	Your value
Allow IBM Protocol Extension	You must specify whether you will allow the use of the IBM PROTOCOL extension. The extension allows a query-string parameter that specifies whether browser artifact or browser POST is used. For more information, see "SAML 1.1 " on page 7 .	One of the following: • Allow IBM Protocol Extension. (Select the check box.) • Do not allow Protocol Extension. (Clear the check box.)

Table 15. Identity mapping information for identity provider in SAML 1.1 federation

Identity mapping	Description	Your value
Identity mapping options • User JavaScript transformation for identity mapping • Use an external web service for identity mapping	If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account. If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

After you complete the tables, continue with the instructions in Creating and modifying a federation.

SAML 2.0 service provider worksheet

If you are the service provider in the federation and use SAML 2.0, use this worksheet to record your configuration information.

Table 16. Federation protocol

Federation protocol	Description	Your value
Federation name	The name you want to give this federation. The name must not contain any ASCII control characters or special characters except hyphen and underscore.	

Table 16. Federation protocol (continued)

Federation protocol	**Description**	**Your value**
Select the protocol for this federation: • **OpenID Connect** • **SAML 2.0**	The protocol you want to use in the federation.	In these instructions, use **SAML 2.0**.

Table 17. Template

Template	**Description**	**Your value**
Select the template: • **Quick Connect** • **SAML 2.0**	Choose **Quick Connect** to quickly set up an identity provider federation to work with partner templates that can assist with the establishment of federations to well-known partners. Choose **SAML 2.0** to use the full set of configuration options. Because this is SAML 2.0 service provider worksheet, select **SAML 2.0** as the template. The template cannot be changed after a federation is created.	**SAML 2.0**

Table 18. General information

General information	**Description**	**Your value**
Company name	The name of the company that is creating this provider.	
Provider ID	A unique identifier that identifies the provider to its partner provider. The default value is `*point_of_contact_server_URL*/*federation_name*/saml20`.	

Table 18. General information (continued)

General information	**Description**	**Your value**
Role	Your role is either **Identity Provider** or **Service Provider**. An identity provider vouches for the identity of the end user. The Identity Provider authenticates the user and provides an authentication token to the service provider. A service provider provides a service to end users. In most cases, service providers do not authenticate users, but instead request authentication decisions from an identity provider. You cannot change the role after a federation is created.	**Service provider**

Table 19. Point of contact server

Point of contact server	**Description**	**Your value**
Point of contact server URL	The endpoint URL of the point of contact server. The point of contact server is a reverse proxy server that is configured in front of the runtime listening interfaces. The format is `http[s]://hostname[:portnumber]/[junction]/sps`	

Table 20. Profile selection

Profile selection	**Description**	**Your value**
SAML 2.0 profile options: • **Web Browser Single Sign-on** • **Name Identifier Management** • **Single Logout**	The profile for your federation. The **Web Browser Single Sign-on** profile must be selected by default. You cannot clear this selection. For more information about profiles, see "SAML profiles" on page 9.	

Table 21. Single Sign-on settings

Settings	Description	Your value
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect**	The choice of binding depends on the type of messages sent. For example, an authentication request message can be sent from service provider to an identity provider. The response message can be sent from an identity provider to a service provider by using either HTTP POST or HTTP artifact. A pair of partners in a federation does not need to use the same binding.	
The default NameID format	The default format determines processing rules for the NameID value if one of the following items is true: • The format attribute is not set • The format attribute is set to `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`	Choose one of the following formats: • `urn:oasis:names:tc:SAML:2.0:namei format:persistent` • `urn:oasis:names:tc:SAML:1.1:namei format:emailAddress` • `urn:oasis:names:tc:SAML:2.0:namei format:transient`
Enable ECP	Check this check box to enable the ECP profile.	
Require signature on incoming SAML assertions	Specifies that you require your partner to sign SAML assertions. You will validate the signature on the incoming SAML assertions.	
Require outgoing SAML authentication requests to be signed	Specifies that you require your partner to validate the signature on SAML authentication requests. You will sign the outgoing SAML authentication requests.	

Table 22. Name Identifier Management settings

Settings	Description	Your value
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect** • **HTTP SOAP**	The choice of binding depends on the type of messages sent. A pair of partners in a federation does not need to use the same binding.	

Table 22. Name Identifier Management settings (continued)

Settings	Description	Your value
Message signatures Select which outgoing SAML messages require a signature: • **Name identifier management requests** • **Name identifier management responses**	Specifies whether you will sign the outgoing SAML name identifier management requests and responses.	

Table 23. Single logout settings

Settings	Description	Your value
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect** • **HTTP SOAP**	The choice of binding depends on the type of messages sent. A pair of partners in a federation does not need to use the same binding.	
Message signatures Select which outgoing SAML messages require a signature: • **Single logout requests** • **Single logout responses**	Specifies whether you will sign the outgoing SAML logout requests and responses.	

Table 24. Signature options

Signatures	Description	Your value
Certificate database	Select the database where the signing certificate is stored	
Certificate label	Name of the certificate to use for signing.	

Table 24. Signature options (continued)

Signatures	Description	Your value
Include the following KeyInfo elements	Determine which KeyInfo elements to include in the digital signature for a SAML message or assertion. **X509 certificate data** Specify whether you want the BASE64 encoded certificate data to be included with your signature. The default action is to include the X.509 certificate data. **X509 Subject Name** Specify whether you want the subject name to be included with your signature. The default action is to exclude the X.509 subject name. **X509 Subject Key Identifier** Specify whether you want the X.509 subject key identifier to be included with your signature. The default action is to exclude the subject key identifier. **X509 Subject Issuer Details** Specify whether you want the issuer name and the certificate serial number to be included with your signature. The default action is to exclude the X.509 subject issuer details. **Public key** Specify whether you want the public key to be included with your signature. The default action is to exclude the public key.	

Table 25. Encryption options

Signatures	Description	Your value
Certificate database	Select the database where the encryption certificate is stored	
Certificate label	Name of the certificate to use for encryption.	

Table 26. SAML message settings

Message settings	**Description**	**Your value**
Message Lifetime in seconds	An integer value specifying the length of time, in seconds, that a message is valid. The default value is `300`.	
Artifact Lifetime in seconds	The length of time, in seconds, that an artifact is considered valid. This field is only valid when HTTP artifact binding has been enabled. The default value is `120`.	
Session Timeout in seconds	The length of time, in seconds, that the session remains valid. The default value is `7200`.	
Select which outgoing messages require a signature: • **Artifact requests** • **Artifact responses**	Specifies whether you will sign the outgoing SAML artifact requests and responses.	
Message issuer format	Format attribute of the Issuer of the SAML message.	
Message issuer name qualifier	Name qualifier attribute of the Issuer of the SAML message.	

Table 27. Identity mapping settings

Identity mapping	**Description**	**Your value**
Identity mapping options • **Use JavaScript transformation for identity mapping** • **Use an external web service for identity mapping**	If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account. If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

Table 28. SAML Message Extensions

Message Extensions	**Description**	**Your value**
SAML Message Extension options: • No message extensions (default) • Use Javascript to add message extensions	If you configure your federation with a message extension rule, every time a SAML message is written, the rule is invoked in order to gather any extensions which need to be included. The mapping rule is invoked with context information about the federation and partner, as well as the kind of message being sent. The mapping rule context is available in a variable 'context'. For documentation on this object see the on box javadoc for the class **`JSMessageExtensionContext`**.	If Javascript extensions are enabled, a subsequent dialogue allows selection of the mapping rule. Traditional identity mapping rules with the category `SAML_2_0` are filtered from the view, as identity mapping rules are not compatible with extension rules. There is a rule available out of the box, which contains information and examples.

After you complete the tables, continue with the instructions in Creating and modifying a federation.

SAML 2.0 identity provider worksheet

If you are the identity provider in the federation and use SAML 2.0, record your configuration information in the following tables.

Table 29. Federation protocol

Federation protocol	**Description**	**Your value**
Federation name	The name you want to give this federation. The name must not contain any ASCII control characters or special characters except hyphen and underscore.	
Select the protocol for this federation: • **OpenID Connect** • **SAML 2.0**	The protocol you want to use in the federation.	In these instructions, use **SAML 2.0**.

Table 30. Template

Template	**Description**	**Your value**
Select the template: • **Quick Connect** • **SAML 2.0**	Choose **Quick Connect** to quickly set up an identity provider federation to work with partner templates that can assist with the establishment of federations to well-known partners. Choose **SAML 2.0** to use the full set of configuration options. The template cannot be changed after a federation is created.	

Table 31. General information

General information	**Description**	**Your value**
Company name	The name of the company that is creating this provider.	
Provider ID	A unique identifier that identifies the provider to its partner provider. The default value is *`point_of_contact_server_URL/federation_name/`*`saml20`.	
Role	Your role is either **Identity Provider** or **Service Provider**. An identity provider vouches for the identity of the end user. The Identity Provider authenticates the user and provides an authentication token to the service provider. A service provider provides a service to end users. In most cases, service providers do not authenticate users, but instead request authentication decisions from an identity provider. You cannot change the role after a federation is created.	**Identity provider**

Table 32. Point of contact server

Point of contact server	**Description**	**Your value**
Point of contact server URL	The endpoint URL of the point of contact server. The point of contact server is a reverse proxy server that is configured in front of the runtime listening interfaces. The format is `http[s]://`*`hostname`*`[:`*`portnumber`*`]/ [`*`junction`*`]/sps`	

Table 33. Profile selection

Profile selection	**Description**	**Your value**
SAML 2.0 profile options: • **Web Browser Single Sign-on** • **Name Identifier Management** • **Single Logout**	The profile for your federation. The **Web Browser Single Sign-on** profile must be selected by default. You cannot clear this selection. For more information about profiles, see "SAML profiles" on page 9.	

Table 34. Single Sign-on settings

Settings	**Description**	**Your value**
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect**	The choice of binding depends on the type of messages sent. For example, an authentication request message can be sent from a service provider to an identity provider. The response message can be sent from an identity provider to a service provider by using either HTTP POST or HTTP artifact. A pair of partners in a federation does not need to use the same binding.	
The default NameID format	The default format determines processing rules for the NameID value if one of the following items is true: • The format attribute is not set • The format attribute is set to `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`	Choose one of the following formats: • `urn:oasis:names:tc:SAML:2.0:namei format:persistent` • `urn:oasis:names:tc:SAML:1.1:namei format:emailAddress` • `urn:oasis:names:tc:SAML:2.0:namei format:transient`

Table 34. Single Sign-on settings (continued)

Settings	Description	Your value
Amount of time, in seconds, before the issue date that an assertion is considered valid	Specifies that you require your partner to sign SAML validations. You will validate the signature on the incoming SAML assertions.	
Amount of time, in seconds, that the assertion is valid before being issued	Specifies that you require your partner to validate the signature on SAML authentication requests. You will sign the outgoing SAML authentication requests.	
Require consent to federate	Requires the identity provider to present a page to the user verifying the federation request.	
Enable ECP	Check this check box to enable the ECP profile.	
Add Session State Headers	Add or delete a Session State Header. Multiple headers can be added.	Specify the name of the Session State Header that you are adding in the field.
Require signature on incoming SAML assertions	Specifies that you require your partner to sign SAML assertions. You will validate the signature on the incoming SAML assertions.	
Require outgoing SAML authentication requests to be signed	Specifies that you require your partner to validate the signature on SAML authentication requests. You will sign the outgoing SAML authentication requests.	

Table 35. Name Identifier Management settings

Settings	Description	Your value
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect** • **HTTP SOAP**	The choice of binding depends on the type of messages sent. A pair of partners in a federation does not need to use the same binding.	
Message signatures Select which outgoing SAML messages require a signature: • **Name identifier management requests** • **Name identifier management responses**	Specifies whether you will sign the outgoing SAML name identifier management requests and responses.	

Table 36. Single logout settings

Settings	**Description**	**Your value**
Bindings: You can choose one or more binding options. • **HTTP Artifact** • **HTTP POST** • **HTTP Redirect** • **HTTP SOAP**	The choice of binding depends on the type of messages sent. A pair of partners in a federation does not need to use the same binding.	
Message signatures Select which outgoing SAML messages require a signature: • **Single logout requests** • **Single logout responses**	Specifies whether you will sign the outgoing SAML logout requests and responses.	
Exclude session index	Select whether to exclude session index in the single logout request. If this property is selected, the logout request message sent out from this Identity Provider will exclude session index. When the Service Provider receives this logout request, it will log out all the sessions for the current user. The Identity Provider will log out only the current user session locally. This setting is used on the identity provider only.	
Optional attribute-`ResponseLocation`		

Table 37. Signature options

Signatures	**Description**	**Your value**
Certificate database	Select the database where the signing certificate is stored	
Certificate label	Name of the certificate to use for signing.	

Table 37. Signature options (continued)

Signatures	Description	Your value
Include the following KeyInfo elements	Determine which KeyInfo elements to include in the digital signature for a SAML message or assertion. **X509 certificate data** Specify whether you want the BASE64 encoded certificate data to be included with your signature. The default action is to include the X.509 certificate data. **X509 Subject Name** Specify whether you want the subject name to be included with your signature. The default action is to exclude the X.509 subject name. **X509 Subject Key Identifier** Specify whether you want the X.509 subject key identifier to be included with your signature. The default action is to exclude the subject key identifier. **X509 Subject Issuer Details** Specify whether you want the issuer name and the certificate serial number to be included with your signature. The default action is to exclude the X.509 subject issuer details. **Public key** Specify whether you want the public key to be included with your signature. The default action is to exclude the public key.	

Table 38. Encryption options

Signatures	Description	Your value
Certificate database	Select the database where the encryption certificate is stored	
Certificate label	Name of the certificate to use for encryption.	

Table 39. SAML message settings

Message settings	**Description**	**Your value**
Message Lifetime in seconds	An integer value specifying the length of time, in seconds, that a message is valid. The default value is `300`.	
Artifact Lifetime in seconds	The length of time, in seconds, that an artifact is considered valid. This field is only valid when HTTP artifact binding has been enabled. The default value is `120`.	
Session Timeout in seconds	The length of time, in seconds, that the session remains valid. The default value is `7200`.	
Select which outgoing messages require a signature: • **Artifact requests** • **Artifact responses**	Specifies whether you will sign the outgoing SAML artifact requests and responses.	
Message issuer format	Format attribute of the Issuer of the SAML message.	
Message issuer name qualifier	Name qualifier attribute of the Issuer of the SAML message.	

Table 40. Access policy settings

Access Policy	**Description**	**Your value**
Enable access policy	If you configure an identity provider, this setting specifies whether to enable access policy. If you enable access policy, you must select one of the policies that you defined. **Note:** If access policy is enabled on both the federation configuration and the partner configuration, the partner configuration takes effect.	

Table 41. Identity mapping settings

Identity mapping	Description	Your value
Identity mapping options • **Use JavaScript transformation for identity mapping** • **Use an external web service for identity mapping**	If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account. If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

Table 42. SAML Message Extensions

Message Extensions	Description	Your value
SAML Message Extension options: • No message extensions (default) • Use Javascript to add message extensions	If you configure your federation with a message extension rule, every time a SAML message is written, the rule is invoked in order to gather any extensions which need to be included. The mapping rule is invoked with context information about the federation and partner, as well as the kind of message being sent. The mapping rule context is available in a variable 'context'. For documentation on this object see the on box javadoc for the class **`JSMessageExtensionContext`**.	If Javascript extensions are enabled, a subsequent dialogue allows selection of the mapping rule. Traditional identity mapping rules with the category `SAML_2_0` are filtered from the view, as identity mapping rules are not compatible with extension rules. There is a rule available out of the box, which contains information and examples.

After you complete the tables, continue with the instructions in Creating and modifying a federation.

Creating a SAML partner

Create a federation partner by gathering the necessary configuration information for input into the local management interface on the appliance.

To set up a federation, follow these steps:

1. Gather the required data. See "Obtaining federation configuration data from your partner" on page 40.
2. Use the local management interface to configure your partner. See Managing federated partners. This process includes exporting a metadata file for the partner.

Obtaining federation configuration data from your partner

You must obtain configuration information from your partner before you can add that partner to a federation.

The partner can export the federation configuration to a metadata file.

To help you gather the appropriate information from your partner, complete the appropriate worksheet for the role that your partner will have in the federation:

- If you are the identity provider, add a service provider partner. Depending on your service provider partner, use one of the following worksheets:
  - "SAML 1.1 service provider partner worksheet" on page 40
  - "SAML 2.0 service provider partner worksheet" on page 50
- If you are the service provider, add an identity provider partner. Depending on your identity provider partner, use one of the following worksheets:
  - "SAML 1.1 identity provider partner worksheet" on page 45
  - "SAML 2.0 identity provider partner worksheet" on page 53

After gathering the configuration information of your partner, use the local management interface to add the federation partner properties. See Managing federation partners.

SAML 1.1 service provider partner worksheet

If you use SAML 1.1 as an identity provider, you must add a service provider partner to your federation. Some information can be supplied to you in a metadata file, or all of the information can be supplied to you manually.

Use the following worksheet to gather the necessary information from your partner. Modify this worksheet to reflect the specific information that you need from your partner. You must also ask your partner to complete that modified worksheet.

Table 43. Federation to which you are adding a service provider partner in a SAML 1.1 federation

Select Federation	**Description**	**Your value**
Federation name	The name of the federation to which you are adding the partner.	

Table 44. Metadata file from your service provider partner in a SAML 1.1 federation

Import metadata	**Description**	**Your value**
Configure the partner manually	Enter the information of your partner manually in the subsequent windows. See Table 45 on page 41 **Note:** If Configure the partner manually is selected in the Create New Partner window and the Next button is clicked, the user is unable to go back to change the option to add a new partner. To rectify this issue, cancel the widget and start again.	

Table 44. Metadata file from your service provider partner in a SAML 1.1 federation (continued)

Import metadata	**Description**	**Your value**
Upload a partner metadata file	The name and path of the file you obtained from your partner that contains the configuration information of your partner.	

Table 45. Configuring a partner manually. Provide the following information if you selected **Configure the partner manually**

Basic Information	**Description**
Name	Provide a name for the partner.
Enabled	Check this for the partner to be active.
Provider ID	Provide a unique identifier that identifies the provider partner to the federation. The value for this must be a URI.

Table 46. Single Sign-On settings for service provider partner in SAML 1.1 federation

Settings	**Description**	**Your value**
Assertion Consumer Service URL	Provide the Assertion Consumer Service URL for the partner. The value for this must be a URI.	
Use artifact profile for SSO	Check this check box to use the artifact profile for single sign-on.	
Include the following attribute types in the SAML assertions	Provide attribute types in the value text box.	A "*" means include all types. It is selected by default.
Subject confirmation method	There are four subject confirmation methods. It no value is set, this field defaults to **No Subject Confirmation Method.**	

Table 47. Assertion Settings

Configure Security Token	**Description**	**Your value**
Sign SAML Assertions	Enable this checkbox if you want to sign SAML assertions.	
Select the key for signing assertions • Keystore in IBM Security Verify Access key service, where the key is stored • Private key you will use for signing the assertion.	If you choose to sign the assertion signatures, you must select a keystore and a key. **Note:** Create the keystore and key before this task.	• KeyStore • Certificate label

Table 47. Assertion Settings (continued)

Configure Security Token	Description	Your value
Include the X509 certificate data	If you choose to sign the SAML assertion, specify whether you want the BASE64 encoded certificate data to be included with your signature. The default action is to include the X.509 certificate data (**Yes**). Or, you can also choose to exclude the X.509 certificate data (**No**).	
Include the X509 Subject Issuer Details	If you choose to sign the SAML assertion, specify whether you want the issuer name and the certificate serial number to be included with your signature. The default action is to exclude (**No**) the X.509 subject issuer details . Or, you can choose to include the X.509 subject issuer details (**Yes**).	
Include the X509 Subject Name	If you choose to sign the SAML assertion, specify whether you want the subject name to be included with your signature. The default action is to exclude the X.509 subject name (**No**). Or, you can choose to include the X.509 subject name (**Yes**).	
Include the X509 Subject Key Identifier	If you choose to sign the SAML assertion, specify whether you want the X.509 subject key identifier to be included with your signature. The default action is to exclude the subject key identifier (**No**). Or, you can choose to include the X.509 subject key identifier (**Yes**).	

Table 47. Assertion Settings (continued)

Configure Security Token	Description	Your value
Include the Public Key	If you choose to sign the SAML assertion, specify whether you want the public key to be included with your signature. The default action is to exclude the public key (**No**). Or, you can choose to include the public key (**Yes**).	
Use the inclusive Namespaces	If you choose to sign the SAML assertion, you can select to use the InclusiveNamespaces element in the canonicalization of the assertion during signature creation. The default is unchecked.	
Signature Algorithm for signing SAML Messages	Specifies the signature algorithm to use for the transaction. The selected key used to sign the SAML messages must match the option chosen in the drop-down menu to prevent signature failure. Select the signature algorithm from the following options. • RSA-SHA1 • DSA-SHA256 • RSA-SHA512	

Table 48. Validation Settings

Signatures	Description	Your value
Validate Signatures on Artifact Requests	You can validate the SAML message signatures when browser artifact is used. To use this option, select the Validate Signatures check box.	One of the following: • Validate signatures for artifact. (Select check box.) • Do not validate signatures for artifact. (Clear check box.)

Table 48. Validation Settings (continued)

Signatures	**Description**	**Your value**
Select the key for validating artifacts:	If you select to validate messages when browser artifact is used, the same validation key is used to validate them. The key you use is the public key that corresponds to the private key that your partner uses to sign messages. **Note:** If you are importing the data of your partner, the key is supplied in the metadata file. If you are manually entering the data of your partner, be sure that you have obtained the key from your partner. Then import the key into the appropriate keystore in the IBM Security Verify Access key service before this task.	• Certificate database • Certificate Label
Signature Algorithm for validating artifacts	Specifies the signature algorithm to use for the transaction. The selected key used to sign the SAML messages must match the option chosen in the drop-down menu to prevent signature failure. Select the signature algorithm from the following options. • RSA-SHA1 • DSA-SHA256 • RSA-SHA512	

Table 49. Identity mapping information for service provider partner in SAML 1.1 federation

Identity mapping	**Description**	**Your value**
Identity mapping options One of the following: • Use the identity mapping that is configured for this partner's federation. • Use JavaScript transformation for identity mapping • Use an external web service for identity mapping	The type of identity mapping to use with this partner. You can choose to use the identity mapping that is configured for this partner's federation. Or, you can choose to override the identity mapping that is configured for this partner's federation.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

SAML 1.1 identity provider partner worksheet

If you use SAML 1.1 as a service provider, you must add an identity provider partner to your federation. Some information can be supplied to you in a metadata file, or all of the information can be supplied to you manually.

Use the following worksheet to gather the necessary information from your partner. Modify this worksheet to reflect the specific information that you need from your partner. You must also ask your partner to complete the modified worksheet.

Table 50. Federation to which you are adding a service provider partner in a SAML 1.1 federation

Select Federation	**Description**	**Your value**
Federation name	The name of the federation to which you are adding the partner.	

Table 51. Metadata file from your service provider partner in a SAML 1.1 federation

Import metadata	Description	Your value
Configure the partner manually	Enter the information of your partner manually in the subsequent windows. See Table 52 on page 46 **Note:** If Configure the partner manually is selected in the Create New Partner window and the Next button is clicked, the user is unable to go back to change the option to add a new partner. To rectify this issue, cancel the widget and start again.	
Metadata file	The name and path of the file you obtained from your partner that contains the configuration information of your partner.	

Table 52. Configuring a partner manually. Provide the following information if you selected **Configure the partner manually**

Basic Information	Description
Name	Provide a name for the partner.
Enabled	Check this for the partner to be active.
Provider ID	Provide a unique identifier that identifies the provider partner to the federation.

Table 53. Single Sign-On Settings

Settings	Description
Artifact Resolution Service URL	The value for this must be a URI.
Intersite Transfer Service URL	The value for this must be a URI.
Create multiple attribute statements in the Universal User	Select this check box to keep multiple attribute statements in the groups they were received in. This option might be necessary if your custom identity mapping rules are written to operate on one or more specific groups of attribute statements. If this check box is not selected, multiple attribute statements are arranged into a single group (AttributeList) in the STSUniversalUser document. The default setting of the check box is not selected and this setting is appropriate for most configurations.

Table 53. Single Sign-On Settings (continued)

Settings	Description
Maximum request life time (in miliseconds)	Default value: -1, which means the request never expires.

Table 54. Signature validation information for identity provider partner in SAML 1.1 federation

Signature Validation	Description	Your value
Validate Signatures on SAML Messages for Artifact Profile (optional)	You have the option of validating the SAML message signatures when browser artifact is used.	One of the following: • Validate signatures for artifact. (Select check box.) • Do not validate signatures for artifact. (Clear check box.)
Select the key for validating artifacts:	If you select to validate messages when browser artifact is used, the same validation key is used to validate them. The key you use is the public key that corresponds to the private key that your partner uses to sign messages. **Note:** If you are importing the data of your partner, the key is supplied in the metadata file. If you are manually entering the data of your partner, be sure that you have obtained the key from your partner. Then import the key into the appropriate keystore in the IBM Security Verify Access key service before this task.	• Certificate database • Certificate Label

Table 55. Assertion Settings

Configure Security Token	Description	Your value
Enable Signature Validation	If your partner signs assertions, you can choose to validate those signatures. In some cases, your partner require you to validate the signatures.	One of the following: • Enable validation signatures. (Select check box.) • Do not validate signatures. (Clear check box.)
Select Validation Key	Specify the type of signature validation to use. • If you select keystore alias, provide the values for certificate keystore and label. • If you select KeyInfo, provide the regular expression that matches the validation key.	One of the following: • Use the KeyInfo of the XML signature to find X.509 certificate for signature validation • Use keystore alias to find public key for signature validation (Default).

Table 55. Assertion Settings (continued)

Configure Security Token	Description	Your value
Select key and truststore • Truststore in IBM Security Verify Access key service, where the key is stored • Public key to use for validating the signature	If you choose to validate the assertion signatures or your partner requires signature validation, you must select a keystore and a key. **Note:** The key you use must be the public key that corresponds to the private key that your partner uses to sign the assertions. Obtain this key and create the keystore before this task.	• Keystore • Certificate label

Table 56. Server certificate validation for your identity provider partner in a SAML 1.1 federation

Server Certificate Validation for SOAP	Description	Your value
Select Server Validation Certificate	The public key for the certificate that shows during SSL communication with your partner. You and your partner must agree on which certificate to use. You must have already obtained the certificate and keystore for the certificate. No password is required. This is a mandatory configuration for browser artifact profile.	• Keystore name • Certificate Label **Note:** If no option is selected, the server certificate validation is disabled.

Table 57. Client authentication for SOAP for your identity provider partner in a SAML 1.1 federation

Client Authentication for SOAP	Description	Your value

Client authentication information
Either:
• **Basic authentication**
  – Username
  – Password
• **Client certificate authentication**
  – Certificate you must present to the server of the identity provider.
  The certificate that you and your identity provider partner agreed that you would present.
  – Keystore in IBM Security Verify Access key service, where the key is stored
• **None**- Client authentication information is disabled.
	If your partner requires mutual authentication, you must know which type you must use.
	If it is basic authentication, you need a user name and password.
	If it is client certificate authentication, you need the certificate that you and your partner have agreed to use.
	Note:
	• If you need a certificate, be sure that you have agreed with your partner where to get it. Then, import it into the appropriate keystore in the IBM Security Verify Access key service before this task.
	• Client certificate authentication does not require a password for the truststore.
		One of the following:
		• Basic authentication information:
		  – Username:
		  – Password:
		• Client certificate authentication information:
		  – Keystore name:
		  – Certificate Label

Table 58. Identity mapping information for service provider partner in SAML 1.1 federation

Identity mapping	Description	Your value
Identity mapping options One of the following: • Use the identity mapping that is configured for this partner's federation. • Use JavaScript transformation for identity mapping • Use an external web service for identity mapping	The type of identity mapping to use with this partner. You can choose to use the identity mapping that is configured for this partner's federation. Or, you can choose to override the identity mapping that is configured for this partner's federation.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

SAML 2.0 service provider partner worksheet

If you use SAML 2.0 in your role as an identity provider, you must add a service provider partner to your federation.

Use the following worksheet to gather the necessary information from your partner. Modify this worksheet to reflect the specific information that you need from your partner and ask your partner to complete that modified worksheet.

Note: If your service provider (SP) partner supports multiple assertion consumer (ACS) service endpoints, the SAML2 identity provider supports multiple ACS endpoints for the partner, in a SP-initiated single sign-on flow from that SP partner. The support is effective once you add the SP partner into the SAML2 identity provider federation. Depending on the ACS URL that is specified in the authentication request message, the identity provider processes it as needed.

Table 59. Federation to which you are adding a service provider partner in a SAML 2.0 federation

Select Federation	Description	Your value
Federation name	The name of the federation to which you are adding the partner.	

Table 60. Metadata file from your service provider partner in a SAML 2.0 federation

Import metadata	Description	Your value
Metadata file	The name and path of the file you obtained from your partner that contains the configuration information of your partner.	

Table 61. Single sign-on settings

Single sign-on settings	Description	Your value
Provide the details for the SAML 2.0 Web Browser Single Sign-On profile	Specify the details for the SAML 2.0 Web Browser Single Sign-On profile. Multiple profiles can be added.	Specify the binding type and URL for the profile that you are adding.
Include the following attributes in the SAML assertions	Specify the attributes to include in the assertion. The source attributes must be created first.	
Include the following attribute types in the SAML assertions (a "*" means include all types)	Specify the types of attributes to include in the assertion. The asterisk (*), which is the default setting, indicates that all of the attribute types will be included in the assertion.	
Amount of time, in seconds, that an idle session for the partner remains valid	Amount of time, in seconds, that an idle session for the partner remains valid. The default value is `3600` seconds.	

Table 61. Single sign-on settings (continued)

Single sign-on settings	Description	Your value
Include federation ID when performing alias service operations.	Indicates whether the key for indexing into the alias service combines the federation ID with the partner Provider ID when performing alias service operations. This feature is useful in scenarios where two or more federations, that use persistent name identifiers, import the same partner metadata.	

Table 62. Server certificate validation for your service provider partner in a SAML 2.0 federation.

Note: Provide the SOAP SSL connection parameters, only if SOAP endpoint is `https`.

SSL server validation for SOAP endpoints	Description	Your value
Select Server Validation Certificate	The public key for the certificate that shows during SSL communication with your partner. You and your partner must agree which certificate to use. You must have already obtained the certificate and added it to your truststore.	
Certificate database	Select the database where the certificate is stored.	
Certificate label	Name of the certificate to use for server validation. If not provided, all certificates in the specified certificate database will be trusted.	

Table 63. Client authentication for your service provider partner in a SAML 2.0 federation

SSL Client Authentication for SOAP endpoints	Description	Your value
Client authentication information • **No authentication** • **Basic authentication** – Username – Password • **Client certificate authentication** – Certificate to present to the server of the identity provider. This certificate is the certificate that you and your identity provider partner agreed to present.	If your partner requires mutual authentication, you must know which type to use. Select **No authentication** if your partner does not require authentication. If it is basic authentication, you need a user name and password. If it is client certificate authentication, you need the certificate that you and your partner have agreed to use. **Note:** If you need a certificate, be sure that you have agreed with your partner where it comes from. Obtain and import it into the appropriate keystore.	One of the following options: • No authentication • Basic authentication information: – Username: – Password: • Client certificate authentication information: – Certificate database – Certificate label

Table 64. Access policy settings

Access Policy	Description	Your value
Enable access policy	Specifies whether to enable access policy. If you enable access policy, you must select one of the policies that you defined. **Note:** If access policy is enabled on both the federation configuration and the partner configuration, the partner configuration takes effect.	

Table 65. Identity Mapping options for your service provider partner in a SAML 2.0 federation

Identity Mapping Options	**Description**	**Your value**
Identity mapping options • Use the identity mapping that is configured for this partner's federation. • Use JavaScript transformation for identity mapping • Use an external web service for identity mapping	The type of identity mapping to use with this partner. You can choose to use the identity mapping that is configured for this partner's federation. Or, you can choose to override the identity mapping that is configured for this partner's federation.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust .

Table 66. SAML Message Extensions

Message Extensions	**Description**	**Your value**
SAML Message Extension options: • No message extensions (default) • Use Javascript to add message extensions • Use the federation configurations (Partner only)	If you configure your federation with a message extension rule, every time a SAML message is written, the rule is invoked in order to gather any extensions which need to be included. The mapping rule is invoked with context information about the federation and partner, as well as the kind of message being sent. The mapping rule context is available in a variable 'context'. For documentation on this object see the on box javadoc for the class **`JSMessageExtensionContext`**.	If Javascript extensions are enabled, a subsequent dialogue allows selection of the mapping rule. Traditional identity mapping rules with the category `SAML_2_0` are filtered from the view, as identity mapping rules are not compatible with extension rules. There is a rule available out of the box, which contains information and examples.

After you complete this worksheet, continue with the steps in Managing federation partners.

SAML 2.0 identity provider partner worksheet

If you use SAML 2.0 in your role as a service provider, you must add an identity provider partner to your federation.

Use the following worksheet to gather the necessary information from your partner. Modify this worksheet to reflect the specific information that you need from your partner and ask your partner to complete that modified worksheet.

Table 67. Federation to which you are adding an identity provider partner in a SAML 2.0 federation

Select Federation	Description	Your value
Federation name	The name of the federation to which you are adding the partner.	

Table 68. Metadata file from your identity provider partner in a SAML 2.0 federation

Import metadata	Description	Your value
Metadata file	The name and path of the file you obtained from your partner that has their configuration information.	

Table 69. Single sign-on settings

Single sign-on settings	Description	Your value
Provide the details for the SAML 2.0 Web Browser Single Sign-On profile	Specify the details for the SAML 2.0 Web Browser Single Sign-On profile. Multiple profiles can be added.	Specify the details of the profile that you are uploading according to the options available.
Include the following attributes in the SAML assertions	Specify the attributes to include in the STSUniversalUser. The source attributes must be created first.	
Force authentication to achieve account linkage	Specify if a user is forced to authenticate at the service provider to perform account linkage. This event occurs if a SAML response is received with an unknown alias in the service provider.	
Include federation ID when performing alias service operations	Indicates whether the key for indexing into the alias service combines the federation ID with the partner Provider ID when performing alias service operations. This feature is useful in scenarios where two or more federations, that use persistent name identifiers, import the same partner metadata.	

Table 69. Single sign-on settings (continued)

Single sign-on settings	Description	Your value
Username to be used for anonymous users	Use this name identifier to access a service through an anonymous identity. The user name entered here is one that the service provider recognizes as a one-time name identifier for a legitimate user in the local user registry. This feature gives users access to a resource on the service provider without establishing a federated identity. This feature is useful in scenarios where the service provider does not need to know the identity of the user account but must only know that the identity provider has authenticated (and can vouch for) the user.	
Map unknown name identifiers to the anonymous username	Specifies that the service provider can map an unknown persistent name identifier alias to the anonymous user account.	
Create multiple attribute statements in the Universal User	Select this check box to keep multiple attribute statements in the groups they were received in. This option might be necessary if your custom identity mapping rules are written to operate on one or more specific groups of attribute statements. If this check box is not selected, multiple attribute statements are arranged into a single group (AttributeList) in the STSUniversalUser document.	

Table 70. Server certificate validation.

Note: Provide the SOAP SSL connection parameters, only if SOAP endpoint is `https`.

SSL server validation for SOAP endpoints	Description	Your value
Select Server Validation Certificate	The public key for the certificate that shows during SSL communication with your partner. You and your partner must agree which certificate to use. You must have already obtained the certificate and added it to your truststore.	
Certificate database	Select the database where the certificate is stored.	
Certificate label	Name of the certificate to use for server validation. If not provided, all certificates in the specified certificate database will be trusted.	

Table 71. Client authentication

SSL Client Authentication for SOAP endpoints	Description	Your value
Client authentication information • **No authentication** • **Basic authentication** – Username – Password • **Client certificate authentication** – Certificate to present to the server of the identity provider. This certificate is the certificate that you and your identity provider partner agreed to present.	If your partner requires mutual authentication, you must know which type to use. Select **No authentication** if your partner does not require authentication. If it is basic authentication, you need a user name and password. If it is client certificate authentication, you need the certificate that you and your partner have agreed to use. **Note:** If you need a certificate, be sure that you have agreed with your partner where it comes from. Obtain and import it into the appropriate keystore.	One of the following options: • No authentication • Basic authentication information: – Username: – Password: • Client certificate authentication information: – Certificate database – Certificate label

Table 72. Identity Mapping

Identity Mapping Options	Description	Your value
Identity mapping options • Use the identity mapping that is configured for this partner's federation. • Use JavaScript transformation for identity mapping • Use an external web service for identity mapping	The type of identity mapping to use with this partner. You can choose to use the identity mapping that is configured for this partner's federation. Or, you can choose to override the identity mapping that is configured for this partner's federation.	If you choose JavaScript for mapping, on a subsequent panel, you are asked to select the JavaScript file to use. If you choose an external web service, on a subsequent panel, you are asked to provide the following information: • URI format (HTTP or HTTPS) • Web service URI • Server Certificate database, if the URI format is HTTPS • Client authentication type, if the URI format is HTTPS • Message format: – XML – WS-Trust

Table 73. SAML Message Extensions

Message Extensions	Description	Your value
SAML Message Extension options: • No message extensions (default) • Use Javascript to add message extensions • Use the federation configurations (Partner only)	If you configure your federation with a message extension rule, every time a SAML message is written, the rule is invoked in order to gather any extensions which need to be included. The mapping rule is invoked with context information about the federation and partner, as well as the kind of message being sent. The mapping rule context is available in a variable ‘context’. For documentation on this object see the on box javadoc for the class **`JSMessageExtensionContext`**.	If Javascript extensions are enabled, a subsequent dialogue allows selection of the mapping rule. Traditional identity mapping rules with the category `SAML_2_0` are filtered from the view, as identity mapping rules are not compatible with extension rules. There is a rule available out of the box, which contains information and examples.

After you complete this worksheet, continue with the steps in Managing federation partners.

SAML 2.0 bindings

SAML requestors and responders communicate by exchanging messages. The mechanism to transport these messages is called a *SAML binding*.

Security Verify Access supports the following bindings:

HTTP redirect
HTTP redirect enables SAML protocol messages to be transmitted within URL parameters. It enables SAML requestors and responders to communicate by using an HTTP user agent as an intermediary.

The intermediary might be necessary if the communicating entities do not have a direct path of communication. The intermediary might also be necessary if the responder requires interaction with a user agent, such as an authentication agent.

HTTP redirect is sometimes called browser redirect in single sign-on operations. This profile is selected by default.

HTTP POST

HTTP POST enables SAML protocol messages to be transmitted within an HTML form by using base64-encoded content. It enables SAML requestors and responders to communicate by using an HTTP user agent as an intermediary.

The agent might be necessary if the communicating entities do not have a direct path of communication. The intermediary might also be necessary if the responder requires interaction with a user agent such as an authentication agent.

HTTP POST is sometimes called Browser POST, particularly when used in single sign-on operations. It uses a self-posting form during the establishment and use of a trusted session between an identity provider, a service provider, and a client (browser).

HTTP artifact

HTTP artifact is a binding in which a SAML request or response (or both) is transmitted by reference by using a unique identifier that is called an artifact.

A separate binding, such as a SOAP binding, is used to exchange the artifact for the actual protocol message. It enables SAML requestors and responders to communicate by using an HTTP user agent as an intermediary.

This setting is used when it is not preferable to expose the message content to the intermediary.

HTTP artifact is sometimes called browser artifact, particularly when used in single sign-on operations. The HTTP artifact uses a SOAP back channel. The SOAP back channel is used to exchange an artifact during the establishment and use of a trusted session between an identity provider, a service provider, and a client (browser).

SOAP

SOAP is a binding that uses Simple Object Access Protocol (SOAP) for communication.

To use SOAP binding, SAML requestors must have a direct communication path with SAML responders.

The choice of binding you have depends on the profile you choose to use in your federation.

SAML 2.0 name identifier formats

SAML 2.0 name identifier formats control how the users at identity providers are mapped to users at service providers during single sign-on.

Security Verify Access supports the following name identifier formats:

Email address

Use the email address name identifier format if you want a user to log in at the service provider as the same user that they use to log in at the identity provider.

For example, if a user is logged in at the identity provider as `user1`, then they will also be logged in as `user1` at the service provider after single sign-on.

Persistent aliases

Use the persistent name identifier format if you want a user to log in at the identity provider as one user, but log in at the service provider as a different user.

Before you can use this name identifier format, you must link the user at the identity provider with the user at the service provider. You can choose to have the user linking done during single sign-on or by using the alias service.

For example, suppose `user1` in the identity provider is linked with `user2` in the service provider. If `user1` is logged in at the identity provider, then they will be logged in as `user2` in service provider after single sign-on.

Transient aliases

Use the transient name identifier format if you want a user to log in as a shared anonymous user, regardless of which user that they use to log in at the identity provider.

For example, suppose `user1` is a shared anonymous user in the service provider. If the user is logged in as `user2` in the identity provider, then they will be logged in as `user1` in the service provider after single sign-on. Similarly, if the user is logged in as `user3` in the identity provider, then they will be logged in also as `user1` in the service provider.

See Alias service for information about how to manage aliases.

Alias service

To manage the aliases, the Federation module uses an *alias service*. The alias service stores and retrieves aliases that are related to a federated identity.

Persistent name identifier format allows you to link a user at the identity provider with a user at the service provider. Security Verify Access stores these account linkages in a high-volume database or an LDAP database. You can manage these account linkages using the alias service REST API. See to the REST API documentation for more information.

Configuring an LDAP alias service database

If you install IBM Security Verify Access, the high-volume database is used to store the alias information by default. However, now the LDAP database can be used to store alias information.

About this task

The alias service manages aliases by accessing an LDAP user registry. The alias service must know information about the LDAP environment that it operates in.

This topic describes the properties that you must specify.

Procedure

Identify the LDAP environment properties.

Table 74. LDAP environment properties

Property	Description
LDAP Server Connection	Specifies the LDAP Server connection name from the list of available LDAP server connection on the appliance. An LDAP server connection can be configured on the appliance by navigating to **Federations** > **Global** > **Server Connection** and adding a server of the type LDAP. **Note:** When you are configuring LDAP settings in **Server Connection** , ensure that the pool size is set in the tuning parameters. Do **not** leave it as default. See "Managing server connections" on page 282.
LDAP BaseDN	The LDAP search string to search the user and store the user alias.

Modifying alias service settings to LDAP

Learn about modifying the setting for your name identifier database

Procedure

1. Log in to the IBM Security Verify Access local management interface.
2. Click **Federations** > **Manage** > **Alias Service Settings**.
3. Select **LDAP**.
4. Select the LDAP Server Connection from the drop-down list of available server connections.
5. Provide the LDAP BaseDN information.
6. Click **Save**.
7. Deploy the pending changes.

Customizing the SAML 2.0 login form

An identity provider can customize the default authentication login page with more contextual information.

When a user requests access to a single sign-on federation, the identity provider initiates single sign-on by authenticating the user. To authenticate the user, the identity provider uses a point of contact server to display a forms-based login page.

When an identity provider participates in multiple federations or hosts multiple partners in one federation, an administrator might want to customize the default login form.

To specify the contextual information to pass to the web reverse proxy login page, use the local management interface to update the Point of Contact profile. In the profile, edit the `authentication.macros` callback parameter. The value of this callback parameter is a list of comma-separated macros. Each macro represents a piece of contextual information.

To identity the macros you want to use, and to review the Point of Contact profile and its callback parameters, see:

- "Supported macros for customizing an authentication login form" on page 60.
- "Callback parameters and values" on page 288.
- "Updating or viewing a point of contact profile" on page 287.

Supported macros for customizing an authentication login form

You can customize an authentication login form with a set macros that are supported by SAML 2.0.

Security Verify Access supplies contextual authentication parameters so that you can customize login forms. The contextual authentication parameters are passed to the web reverse proxy as query string parameters.

Table 75 on page 60 shows the list of macros names and the name of the query string parameter in which the contextual information is passed to the web reverse proxy login page.

Specify a list of these comma-separated macros in the **`authentication.macros`** callback parameter. See Callback parameters and values.

Table 75. Macros for customizing the login form

Macro	Query-String Parameter name	Description
`%FEDID%`	FedId	Specifies the unique identifier of the federation.
`%FEDNAME%`	FedName	Specifies the user-assigned name of the federation.

Table 75. Macros for customizing the login form (continued)

Macro	Query-String Parameter name	Description
%PARTNERID%	PartnerId	Specifies the provider ID of the partner.
%TARGET%	Target	Specifies the target URL at the partner.
%SPRELAYSTATE%	SPRelayState	Specifies the RelayState data that accompanies the SAML authentication request.
%ACSURL%	AssertionConsumerURL	Specifies the assertion consumer service URL of the partner.
%AUTHNCONTEXT%	AuthnContext	Specifies the RequestedAuthnContext in the SAML authentication request.
%SSOREQUEST%	SSORequest	Specifies the base-64 encoded form of SAML authentication request.
%FORCEAUTHN%	ForceAuthn	Specifies ForceAuthn in SAML authentication request.

Customizing AuthnContext using identity mapping rule

SAML 2.0 Identity Provider now supports customizing `AuthnContextClassRef` using the mapping rule.

The `AuthnContextClassRef`, `AuthnContextDeclRef`, and `AuthnContextComparison` are retrieved from the **Authentication Request**.

The STSUUSER method *addContextAttribute* can be used to set `AuthnContextClassRef` to a required value. In the example below, the `AuthnContextClassRef` is set to `urn:oasis:names:tc:SAML:2.0:ac:classes:X509`.

```
stsuu.addContextAttribute(new Attribute("AssertionAuthnContextRef",
"urn:oasis:names:tc:SAML:2.0:assertion",
"urn:oasis:names:tc:SAML:2.0:ac:classes:X509"));
```

The `AuthnContextComparison` value is available at the mapping rule. The administrator can write logic in the mapping rule to decide on what “exact”, “better”, “minimum”, or “maximum” represents. The administrator can then decide the `AuthnContextClassRef` that needs to be sent in the SAML response.

A sample mapping rule `saml20_authncontext.js` is provided with samples of how these parameters is used. From the dashboard, the mapping rule is under **File Downloads** > **federation** > **examples** > **mapping rules**.

Customizing SAML 2.0 pages

Verify Access generates files that are displayed in response to events that occur during single sign-on requests. The response that is displayed might be a form, such as when login information is required, or an error or information statement about a condition that occurred while the request was processed.

You can customize the event pages by modifying their appearance or content.

Before you continue with the customization, you need to have a thorough understanding of how event pages are generated and displayed.

Generation of event pages

Event pages are displayed in response to events that occur during single sign-on requests. They usually contain a form (such as a prompt for user name and password information) or text (such as an informational or error message).

Event pages are dynamic pages that are generated by Security Verify Access by using the following information:

Template files
XML or HTML files that are provided with the appliance and contain elements, such as fields, text, or graphics, and sometimes macros that are replaced with information that is specific to the request or to provide a response to the request.

Page identifiers
Event information that corresponds to one or more template files. Each page identifier corresponds to a specific event condition, such as a specific error or a condition in which a message or a form must be displayed.

Message catalogs
Text that is used to replace macros in the template files.

When a request is received, the appropriate response page is generated as follows:

1. Processing of the request occurs and a response to an event is required.
2. Template files and page identifiers are read from the file system.
3. Macros in the template files are replaced with values that are appropriate for the response that is needed.
4. An appropriate event page is generated.
5. The generated event page is displayed.

SAML 2.0 page identifiers

The SAML 2.0 runtime can display HTML pages in response to events that occur during single sign-on requests. You can select which pages to display and also modify the pages.

Use HTML pages for the following purposes:

- Displaying success and error messages to users
- Asking users for confirmation
- Sending SAML messages

You can customize these HTML pages so that they display what you want. These pages contain *macros* and are similar to other HTML pages in Security Verify Access. A macro is text in an HTML page that is replaced with context-specific information. For example, the macro `@ERROR_MESSSAGE@` is replaced by text that describes the error that occurred.

You can find the SAML 2.0 pages in the local management interface using these steps:

1. Click **Federation** > **Global Settings** > **Template Files**.
2. Expand the locale folder to locate a template file.

For example, the English version of the SAML `consent_to_federate.html` template is in `C/saml20`.

All of the available SAML 2.0 HTML pages are listed in the following table.

Table 76. SAML 2.0 HTML page identifiers and macros

Page identifier	Description	Macros and descriptions
`saml20/consent_to_federate.html`	Displays during the SAML single sign-on flow whenever the service provider wants to federate the account at the identity provider with the account at the service provider.	**`@TOKEN:form_action@`** The URL to which the SAML message is sent. **`@TOKEN:SPProviderID@`** The ID of the Service Provider. **`@TOKEN:SPDisplayName@`** The name of the Service Provider. **`@TOKEN:IPProviderID@`** The name of the Identity Provider.
`saml20/logout_partial_success.html`	Displays whenever the SAML single log out flow completes with partial success.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation.
`saml20/logout_success.html`	Displays whenever the SAML single log out flow completes successfully.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation.
`saml20/nimgmt_terminate_success.html`	Displays whenever the SAML name identifier management terminate flow completes successfully.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation. **`@TOKEN:PartnerID@`** The ID of the partner.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/nimgmt_update_success.html`	Displays whenever the SAML name identifier management update flow completes successfully.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@TOKEN:UserName@** The user name that performs the operation. **@TOKEN:PartnerID@** The ID of the partner.
`saml20/saml_post_artifact.html`	Sends the SAML artifact to the partner for HTTP POST binding.	**@TOKEN:form_action@** The URL to which the SAML message is sent. **@TOKEN:RelayState@** The RelayState. **@TOKEN:SamlMessage@** The SAML message.
`saml20/saml_post_request.html`	Sends the SAML request message to partner for HTTP POST binding.	**@TOKEN:form_action@** The URL to which the SAML message is sent. **@TOKEN:RelayState@** The RelayState. **@TOKEN:SamlMessage@** The SAML message.
`saml20/saml_post_response.html`	Sends the SAML response message to the partner for HTTP POST binding.	**@TOKEN:form_action@** The URL to which the SAML message is sent. **@TOKEN:RelayState@** The RelayState. **@TOKEN:SamlMessage@** The SAML message.
`saml20/art_exchange_failed.html`	Displays whenever there is a failure during the SAML artifact resolution flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/authn_failed.html`	Displays whenever there is a failure during the SAML single sign-on flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_building_msg.html`	Displays whenever an outgoing SAML message is not constructed.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_decrypting_msg.html`	Displays whenever an incoming SAML message is decrypted.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_missing_config_param.html`	Displays whenever a SAML flow is run on a SAML federation with invalid configuration.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	**Description**	**Macros and descriptions**
`saml20/error_parsing_art.html`	Displays whenever an incoming SAML artifact is parsed.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_parsing_msg.html`	Displays whenever an incoming SAML message is parsed.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_sending_msg.html`	Displays whenever an outgoing SAML message is sent.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_validating_art.html`	Displays whenever an incoming SAML artifact is validated.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/error_validating_init_msg.html`	Displays whenever a SAML flow is initiated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_validating_msg.html`	Displays whenever an incoming SAML message is validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_validating_msg_signature.html`	Displays whenever an incoming SAML message is signature validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/invalid_art.html`	Displays whenever an incoming SAML artifact is validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/invalid_init_msg.html`	Displays whenever a SAML flow is initiated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/invalid_msg.html`	Displays whenever an incoming SAML message is validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/logout_failed.html`	Displays whenever there is a failure during SAML single logout flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/nimgmt_terminate_failed.html`	Displays whenever there is a failure during the SAML name identifier terminate management flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 76. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/nimgmt_update_failed.html`	Displays whenever there is a failure during the SAML name identifier update management flow.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.

Template page for the WAYF page

The Where Are You From (WAYF) page is used at the service provider. The WAYF page enables users to select their identity provider if there is more than one configured in the federation.

When a user arrives at a service provider, a WAYF identifier can be delivered through a cookie or query-string parameter with the request. The entity ID of the identity provider is stored as the value of the cookie or query-string parameter. If the WAYF identifier cookie or query-string parameter is not present, the WAYF page opens.

An example URL that includes the query string parameter for WAYF:

```
https://sp.host.com/isam/sps/samlfed/saml20/
logininitial?RequestBinding=HTTPRedirect&ResponseBinding
=HTTPPost&ITFIM_WAYF_IDP=https://idp.host.com/isam/sps/samlfed/saml20
```

This example is for a SAML 2.0 single sign-on URL. The query string parameter name is `ITFIM_WAYF_IDP`. The value of the identity provider ID is `https://idp.host.com/isam/sps/samlfed/saml20`.

The WAYF page requires the user to indicate where they came from. If the user is not logged on to their identity provider, they are asked to log on. Depending on the attributes passed, the service provider can grant or deny access to the service.

You can find the template pages for WAYF in the local management interface using these steps:

1. Click **Federation** > **Global Settings** > **Template Files**.
2. Expand the locale folder and navigate to `/pages/itfim/wayf`.

Administrators can use the WAYF page without modifications, but in some cases might want to modify the HTML style to match the specific deployment environment.

This template file provides several replacement macros:

@WAYF_FORM_ACTION@
This macro is replaced with the endpoint of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_METHOD@
This macro is replaced with the HTTP method of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_PARAM_ID@
This macro is replaced with ID used by the action for the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_ID@
This macro is replaced with the unique ID of the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_DISPLAY_NAME@
This macro is replaced with the configured display name of the identity provider. This macro is repeated once for each identity provider.

@WAYF_HIDDEN_NAME@
This macro is replaced with the name of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

@WAYF_HIDDEN_VALUE@
This macro is replaced with the value of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

Customizing the Consent to Federate Page

A *consent to federate page* is an HTML form which prompts a user to give consent to joining a federation. You can customize the *consent to federate page* to specify what information it requests from a user.

Before you begin

Determine what values you want to use for the consent to federate page.

About this task

When a user accesses a federation, they agree to join the federation. The HTML form `saml20/consent_to_federate.html` prompts for this consent. You can customize what the form requests by adding consent values. These values indicate how a user agrees to join a federation and if service providers are notified of the consent. Identity providers receive the consent values in the SAML 2.0 response.

The following values determine how a user joins a federation:

1
A user agrees to join a federation without notifying the service provider.

0
A user refuses to join a federation.

A URI value
A URI can indicate whether the user agrees to join a federation and if you want to notify the service provider about the user consent. The following table lists and describes the supported URI values.

Table 77. Supported consent values for SAML 2.0 response

Consent value	URI	Description
Unspecified	`urn:oasis:names:tc:SAML:2.0:consent:unspecified`	The consent of the user is not specified.
Obtained	`urn:oasis:names:tc:SAML:2.0:consent: obtained`	Specifies that user consent is acquired by the issuer of the message.
Prior	`urn:oasis:names:tc:SAML:2.0:consent: prior`	Specifies that user consent is acquired by the issuer of the message before the action which initiated the message.

Table 77. Supported consent values for SAML 2.0 response (continued)

Consent value	URI	Description
Implicit	`urn:oasis:names:tc: SAML:2.0:consent: current-implicit`	Specifies that user consent is implicitly acquired by the issuer of the message when the message was initiated.
Explicit	`urn:oasis:names:tc: SAML:2.0:consent: current-explicit`	Specifies that the user consent is explicitly acquired by the issuer of the message at the instance that the message was sent.
Unavailable	`urn:oasis:names:tc: SAML:2.0:consent: unavailable`	Specifies that the issuer of the message was not able to get consent from the user.
Inapplicable	`urn:oasis:names:tc: SAML:2.0:consent: inapplicable`	Specifies that the issuer of the message does not need to get or report the user consent.

Follow the steps in this procedure to customize the consent to federate page.

Procedure

1. Log in to the local management interface.
2. Click **Federation** > **Global Settings** > **Template Files**.
3. Expand a locale and select `saml20/consent_to_federate.html`.
4. Click **Edit** and add the appropriate consent values for your federation.
5. Click **Save**.
6. Deploy the changes.

Example

The following example shows an added URI with a consent value `Obtained`:

```
<input type="radio" checked name="Consent"
value="urn:urn:oasis:names:tc:SAML:2.0:consent:obtained"/>
Consent Obtained.<br/>
```

In this example, the user consent is acquired by the issuer of the message.

Configuring the user session ID for the federation runtime

Customize the user session ID header name so that you can track user sessions, end sessions, or sign out a particular user from a web reverse proxy point of contact server.

Before you begin

Set your web reverse proxy to enable the creation of unique user session IDs. The following stanza and entry must be set:

```
[session]
user-session-ids = yes
```

See User session management for back-end servers for general information.

About this task

The federation runtime uses user session ID information to log out the user from the web reverse proxy.

The user session ID uniquely identifies a specific session for an authenticated user and is stored as a part of credential information of the user.

The federation runtime obtains the user session ID from the web reverse proxy. The web reverse proxy sends the user session ID to the federation runtime in an HTTP header. To accomplish this, configure the web reverse proxy and federation runtime to use the same header name.

Procedure

1. For the federation runtime, update the advanced configuration property, **`poc.signOut.userSessionRequestHeader`** by using the local management interface:
   a) Select **Federation** > **Global Settings** > **Advanced Configuration**.
   b) Locate **`poc.signOut.userSessionRequestHeader`** in the list, select it, and click **Edit**.
   c) Enter the header name that you want to use for the user session ID and click **Save**.
   For example, specify `my_user_session_id`.

   See "Managing advanced configuration" on page 184.
2. Optional: For the web reverse proxy, update the junction to delete an existing `user_session_id` by using the following command:

   ```
   pdadmin -a sec_master -p password object modify /WebSEAL/fedname-webseal/junction_name
     delete attribute HTTP-Tag-Value user_session_id=user_session_id
   ```

   Where:

   password
   Specifies the password for `sec_master`.

   fedname
   Specifies the name of the federation.

   webseal
   Specifies the name of the web reverse proxy server.

   junction_name
   Specifies the name of the junction.

   user_session_id
   Specifies the existing name that was defined for the session ID.

   For example:

   ```
   pdadmin -a sec_master -p ipadminpw object modify
     /WebSEAL/saml20-ip-ipwga/isam
     delete attribute HTTP-Tag-Value user_session_id=user_session_id
   ```

   This command deletes the existing `user_session_id`.
3. For the web reverse proxy, update the junction to use the **`poc.signOut.userSessionRequestHeader`** property value you defined in step "1" on page 72. Add this customized attribute value by using the following command:

   ```
   pdadmin -a sec_master -p password object modify /fedname-webseal/junction_name
     set attribute HTTP-Tag-Value user_session_id=user_session_ID
   ```

   Where:

   user_session_id
   Specifies the value from step "1" on page 72. For example, `my_user_session_id`.

   For example:

   ```
   pdadmin -a sec_master -p ipadminpw object modify /WebSEAL/saml20-ip-ipwga/isam
     set attribute HTTP-Tag-Value user_session_id=my_user_session_ID
   ```

   This command changes the value for the web reverse proxy to `my_user_session_ID`.

Results

The federation runtime and the web reverse proxy have the same header name.

Synchronizing system clocks in the federation

Because security tokens have expiration times, you and your partner's system clocks must be synchronized.

About this task

In your environment, ensure that the clock on the system where you have the runtime and management services component installed is synchronized with your partner.

See the information of your operating system documentation for information about your system clock and time synchronization. Consider using the NTP time synchronization protocol.

Chapter 4. WS-Federation federations

The Federation Module supports WS-Federation federations.

WS-Federation is a protocol that you can use to accomplish federated single sign-on from identity providers to service providers. In federated single sign-on, users authenticate at identity provider. Service providers use the identity information asserted by identity providers.

WS-Federation protocol defines a standardized, multi-vendor web-based single sign-on solution based on a collection of integrated Web Services (WS*) standards such as WS-Security, WS-Trust, and WS-Federation.

Review the WS-Federation standards documents before you implement a single sign-on federation. The standards specify data exchange and message processing. Understand what information you must provide to your business partners, and what information your partner must provide to you.

WS-Federation single sign-on profiles

The single sign-on profiles enable a client by using a Web browser to achieve single sign-on access to resources within a WS-Federation 1.0 federation.

Typically the user wants to access a resource provided by a service provider, and must authenticate with an identity provider in order to be granted that access.

The profile provides a mechanism for the Web user to obtain an authentication assertion that can be used to establish a security context within the federation. Establishment of the security context enables a user to access multiple resources within the federation without having to authenticate more than once.

WS-Federation support two profiles for use with single sign-on sessions:

`Browser POST`
Browser POST uses a self-posting form during the establishment and use of a trusted session between an identity provider, a service provider, and a client (browser).

WS-Federation supports browser POST by default. No configuration is required.

`Single logout`
This profile terminates all log in sessions within the federation for a specified user. WS-Federation supports single logout by default. No configuration is required.

Identity provider and service provider roles

Each partner in a federation has a role. The role is either Identity Provider or Service Provider. Understand the behavior of each role.

- Identity provider

  An identity provider is a federation partner that vouches for the identity of a user. The Identity Provider authenticates the user, and provides an authentication token to the service provider.

  The identity provider is responsible for the following tasks:

  – Directly authenticates the use by validating a user name and password.
  – Indirectly authenticates the user by validating an assertion about the user's identity as presented by a separate identity provider.

  The identity provider handles the management of user identities to free the service provider from this responsibility.

- Service Provider

  A service provider is a federation partner that provides services to the user. Typically, service providers do not authenticate users, but instead request authentication decisions from an identity provider.

Service providers rely on identity providers to assert the identity of a user, and rely on identity providers to manage user identities for the federation.

Service providers can maintain a local account for the user, which can be referenced by an identifier for the user.

Creating a WS-Federation federation

To create a federation, review the configuration properties, run the configuration wizard, and configure a reverse proxy for the federation.

Before you begin

Ensure that you created a reverse proxy. During federation configuration, you need to specify a reverse proxy URL for the point of contact server. If you need to create a reverse proxy, see Configuring an instance.

Procedure

1. Plan your federation configuration by reviewing the configuration properties. See "WS-Federation federation properties" on page 76.

   Note: Security Verify Access support for WS-Federation includes a customized template for use with Microsoft SharePoint. This template expedites the federation configuration for SharePoint. Deployments with Microsoft SharePoint must use an identity provider federation. For more information, see "WS-Federation federation properties" on page 76.
2. Use the local management interface to create your role in the federation. See Creating and modifying federation properties.
3. Create and configure a reverse proxy instance to act as the point of contact for the federation. See Adding a federation for a reverse proxy server.

What to do next

Next, you can set up your federation partner. See "Creating a WS-Federation partner" on page 78.

WS-Federation federation properties

To configure a WS-Federation federation, you must specify values for a set of properties.

The properties in this list describe the inputs that you must provide when you use the LMI wizard to configure a federation. Most properties are specified for both identity provider and service provider federations. The exceptions are described below.

- Identity provider only
  - **Amount of time, in seconds, before the issue date that an assertion is considered valid**
  - **Amount of time, in seconds, that the assertion is valid before being issued**
- Service provider only
  - **Enable one-time assertion use enforcement**

Federation properties descriptions

Federation name
: The name that you want to give this federation.

  The name must not contain any ASCII control characters or special characters except hyphen and underscore.

Select the protocol for this federation
: WS-Federation

Select the template

Choose **SharePoint** to quickly set up an identity provider federation to work with partner templates that can assist with the establishment of federations to SharePoint partners.

Choose **WS-Federation** to use the full set of configuration options.

Company name

The name of the company that is creating this provider.

Role

Your role is either **Identity Provider** or **Service Provider**.

An identity provider vouches for the identity of the user. The Identity Provider authenticates the user and provides an authentication token to the service provider.

A service provider provides a service to users. In most cases, service providers do not authenticate users, but instead request authentication decisions from an identity provider. You cannot change the role after a federation is created.

Note: When you use the SharePoint template, the **Role** field is not displayed because the **Identity Provider** role is automatically set. SharePoint deployments do not use **Service Provider** federations.

Point of contact server URL

The endpoint URL of the point of contact server. The point of contact server is a reverse proxy server that is configured in front of the runtime listening interfaces. The format is:

```
http[s]://hostname[:portnumber]/[junction]/sps
```

For example, `https://test.com/isam/sps`.

To view your reverse proxy configuration, see Reverse proxy instance management.

Enable one-time assertion use enforcement

Service provider configuration only.

Specifies whether to use the assertion or token only one time. You can select or clear this option.

Amount of time, in seconds, before the issue date that an assertion is considered valid

Identity provider configuration only.

Default value 300 seconds. There is no minimum or maximum enforced.

Amount of time, in seconds, that the assertion is valid before being issued

Identity provider configuration only.

An integer value that specifies the number of seconds that the assertion remains valid. The default value is 300 seconds.

Identity mapping

Identity mapping options

- **Do not perform identity mapping**
- **Use JavaScript transformation for identity mapping**
- **Use an external web service for identity mapping**

If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account.

If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.

If you choose JavaScript for mapping, on a subsequent page, you are asked to select the JavaScript file to use.

If you choose an external web service, on a subsequent page, you are asked to provide the following information:

- URI format (HTTP or HTTPS)
- Web service URI
- Server Certificate database, if the URI format is HTTPS.
- Client authentication type, if the URI format is HTTPS.
- Message format:
  - XML
  - WS-Trust

Creating a WS-Federation partner

Create a federation partner by reviewing the configuration properties and then running the local management interface wizard on the appliance.

Before you begin

You must configure a federation before you create and add a partner. If you did not yet create the federation for this partner, see "Creating a WS-Federation federation" on page 76.

Procedure

1. Plan your federation configuration by reviewing the configuration properties. See "WS-Federation partner properties" on page 78.
2. Use the local management interface to configure your partner. See Managing federated partners.

WS-Federation partner properties

To configure a WS-Federation federation partner, you must specify values for a set of properties.

The properties in this list describe the inputs that you must provide when you use the LMI wizard to configure a partner for a WS-Federation federation. The list consists of three sections:

- Common properties that are used by both identity provider and service provider partners
- Properties that are used by only the identity provider partner
- Properties that are used by only the service provider partner

Be sure to review both the common properties section and the section for your type of partner.

Common properties for both identity provider partners and service provider partners

Federation name
The name of the federation to which you are adding the partner.

Enabled
Specifies whether to enable the partner. Select or clear.

Connection Template
Displays the type of template that is used. The partner wizard automatically detects which template (default or SharePoint) was used to create the federation, and uses the same template to create the partner. The field is read-only.

The name of the WS-Federation realm for this partner

The name of the WS-Federation Realm. This name is the unique identifier for this instance of Security Verify Access. The Realm name is included in assertions that are sent to federation partners. Partners rely on finding a known (defined) Realm name to accept the assertions.

To determine the Realm name, use the local management interface to view the federation configuration. Select **Federation > Manage > Federations**, select your federation, and click **Edit**. On the **Point of Contact Server** pane, make note of the **Realm** value that the wizard displays, and click **Cancel** to exit the wizard. For more information, see Creating and modifying a federation.

The Realm name is generated from the point of contact server value. For example, if the point of contact server URL is `https://test.com/isam/sps` then the realm is set as:

```
https://test.com/isam/sps/wsfed/wsf
```

In the example above, the string `wsfed` is the name of the federation.

The name of the WS-Federation endpoint for this partner

The endpoint for all requests for WS-Federation services. The endpoint is generated from the point of contact server URL value.

To determine the WS-Federation endpoint name, use the local management interface to view the federation configuration. Select **Federation > Manage > Federations**, select your federation, and click **Edit**. On the **Point of Contact Server** pane, make note of the **Endpoint** value that the wizard displays, and click **Cancel** to exit the wizard. For more information, see Creating and modifying a federation.

For example, if the point of contact server URL is `https://test.com/isam/sps` then the endpoint is set to:

```
https://test.com/isam/sps/wsfed/wsf
```

In the example above, the string `wsfed` is the name of the federation.

Maximum request lifetime (in milliseconds)

Amount of time, in milliseconds, that the request is valid. A value of `-1` means that the request lifetime has no limit.

Partner role

Identity Provider or Service Provider. The partner role is read-only and is the opposite of the federation role.

Note: SharePoint partners must be service providers because all SharePoint federations are identity provider federations.

Identity mapping options

- Use the identity mapping that is configured for this partner's federation.
- Do not perform identity mapping.
- Use JavaScript transformation for identity mapping.
- Use an external web service for identity mapping.

The type of identity mapping to use with this partner. You can choose to use the identity mapping that is configured for this partner's federation. Or, you can choose to override the identity mapping that is configured for this partner's federation.

If you choose JavaScript for mapping, on a subsequent page you are asked to select the JavaScript file to use.

If you choose an external web service, on a subsequent page you are asked to provide the following information:

- URI format (HTTP or HTTPS)
- Web service URI
- Server Certificate database, if the URI format is HTTPS.
- Client authentication type, if the URI format is HTTPS.
- Message format:
  - XML
  - WS-Trust

Properties for only the identity provider partner

Create multiple attribute statements in the Universal User

Identity provider partner only.

Select or clear. Select this check box to keep multiple attribute statements in the groups they were received in. This option might be necessary if your custom identity mapping rules are written to operate on one or more specific groups of attribute statements. If this check box is not selected, multiple attribute statements are arranged into a single group (AttributeList) in the `STSUniversalUser` document.

Enable signature validation

Identity provider partner only.

Enables or disables validation of signatures in the token module. Select or clear.

Use the keystore alias to find the public key for signature validation

Identity provider partner only.

Specifies a public key for signature validation, which is the default. Select the certificate database and label.

Certificate database

For identity provider partner.

This property is displayed if you choose to use the keystore alias. Select the certificate database to use for validation.

Certificate label

For identity provider partner.

This property is displayed if you choose to use the keystore alias. Select the certificate label for validation.

Use the KeyInfo of the XML signature to find the X509 Certificate for signature validation

Identity provider partner only.

Determines the appropriate certificate for signature validation. When you select this option, you must provide the subject distinguished name that matches the certificate.

Regexp

Identity provider partner only.

Specifies a regular expression to validate the subject distinguished name that is returned in the`KeyInfo`.

Properties for only the service provider partner

Include the following attribute types in the SAML assertions (a "*" means include all types)

Service provider partner only.

Specifies the types of attributes to be inserted during token creation. The attributes consist of information about the identity (user). Use && to separate attribute types. By default, all types are supported, as indicated by the asterisk (*) wildcard character. For example, to add user-defined attribute types `type1` and `type2`, enter:

```
type1&&type2
```

Subject confirmation method

Service provider partner only.

Specifies the subject confirmation method for the assertion. You can select one confirmation method, or choose `No Subject Confirmation Method`. If you select the holder-of-key type, the default includes the X.509 Certificate Data in the `KeyInfo` for the `SubjectConfirmationMethod`. `STSUniversalUser` can provide the data for the subject confirmation method `KeyInfo`. The data can also be extracted from the signed request data.

Valid values:

- `No Subject Confirmation Method`
- `urn:oasis:names:tc:SAML:1.0:bearer`
- `urn:oasis:names:tc:SAML:1.0:holder-of-key`
- `urn:oasis:names:tc:SAML:1.0:sender-vouches`

Sign SAML assertions
Service provider partner only.

Select if SAML assertions must be signed.

Certificate database
Service provider partner only.

Select the database where the signing certificate is stored.

Certificate label
Service provider partner only.

Name of the certificate to use for signing.

Include the following KeyInfo elements
Service provider partner only.

Determines what `KeyInfo` elements to include in the digital signature for a SAML message or assertion. Select one or more of the following elements.

X509 certificate data
Specify whether you want the BASE64 encoded certificate data to be included with your signature. The default action is to include the X.509 certificate data.

X509 Subject Name
Specify whether you want the subject name to be included with your signature. The default action is to exclude the X.509 subject name.

X509 Subject Key Identifier
Specify whether you want the X.509 subject key identifier to be included with your signature. The default action is to exclude the subject key identifier.

X509 Subject Issuer Details
Specify whether you want the issuer name and the certificate serial number to be included with your signature. The default action is to exclude the X.509 subject issuer details.

Public key
Specify whether you want the public key to be included with your signature. The default action is to exclude the public key.

Note: If you do not select any of the `KeyInfo` elements, X.509 certificate data is still included in the signature by default.

Use Inclusive Namespaces
Service provider partner only.

Specifies whether to use the `InclusiveNamespaces` construct, which means employing exclusive XML canonicalization for greater standardization. The default is cleared.

Signature algorithm for signing SAML assertions
Service provider partner only.

Specifies the signature algorithm to use to sign the SAML assertion.

- RSA-SHA1

```
http://www.w3.org/2000/09/xmldsig#rsa-sha1
```

- RSA-SHA256

  ```
  http://www.w3.org/2001/04/xmldsig-more#rsa-sha256
  ```
- RSA-SHA512

  ```
  http://www.w3.org/2001/04/xmldsig-more#rsa-sha512
  ```

Excluding elements from a WS-FED Request Security Token Response

The default configuration of a Security Verify Access WS-Federation federation specifies a list of elements to exclude from the WS-Federation request security token response (RSTR). This default configuration enables WS-Federation single sign-on to work in the majority of scenarios, such as single sign-on to a Security Verify Access appliance, and single sign-on to a Microsoft SharePoint deployment.

The custom property `wsfed.idp.rstr.excluded.elements` is used to exclude a comma-separated list of elements. The elements that are excluded by default are "Forwardable", "Delegatable", "Status", and "Renewing". The LMI displays the default custom property `wsfed.idp.rstr.excluded.elements` with the following value:

```
default=Forwardable,Delegatable,Status,Renewing
```

Certain applications require a different set of excluded elements. For these cases, you can use the Security Verify Access Advanced Configuration feature to set a custom property to specify the set of elements. You must specify the federation realm for which your set applies. Optionally, you can also set elements of a per-partner basis for the federation.

You can use the following syntax to specify elements are needed:

```
default=<comma_separated_list_of_elements>:<federation_realm>=<comma_separated_list_of_elements>:
            <federation_realm>%<partner_realm>=<comma_separated_list_of_elements>
```

For example, if a federation requires that the only excluded elements are `Forwardable` and `Delegatable`, you can modify the custom property. For this example, to modify the custom property for a federation `fed1` with a realm `fed1-REALM`, set the custom property as follows:

```
default=Forwardable,Delegatable,Status,Renewing:fed1-REALM=Forwardable,Delegatable
```

You can also modify the custom property to allow for requirements specific to a federation partner.

For example, if federation `fed1` from the example above has a partner `partner1` with a realm of `partner1-REALM`, and this partner allows only the `Status` element to be excluded, you can set the custom property `wsfed.idp.rstr.excluded.elements` as follows:

```
default=Forwardable,Delegatable,Status,Renewing:fed1-REALM=Forwardable,Delegatable:
              fed1-REALM%partner1-REALM=Status
```

For information on how to use the LMI Advanced Configuration menu to set custom properties, see "Managing advanced configuration" on page 184.

Chapter 5. OpenID Connect federations

The Federation module supports OpenID Connect (OIDC) Provider federations and OIDC Relying Party federations.

Security Verify Access supports the OAuth 2.0 protocol, including OIDC. The OIDC protocol is an extension of the OAuth protocol to better support identity and authentication. To understand how OIDC extends OAuth, and to understand OIDC Provider federations and Relying Party federations, see:

- OAuth 2.0 and OIDC support
- OpenID Connect concepts
- OAuth 2.0 and OIDC workflows

Versions of Security Verify Access prior to 9.0.4 configured OIDC federations through a federation wizard. Security Verify Access 9.0.4 now configures OIDC Providers through an API Protection interface. Relying Party federations use a new federation wizard that is enhanced to support new capabilities.

Existing deployments of Security Verify Access OIDC federations are fully supported as legacy federations.

For configuration, use the instructions that apply to your deployment:

- For new OIDC Providers, see "OpenID Connect Provider federations" on page 83.
- For new OIDC Relying Party federations, see "OpenID Connect Relying Party federations" on page 83.
- For existing (prior to Security Verify Access 9.0.4) OIDC Provider federations and Relying Party federations, see Legacy support for OpenID Connect federations.

OpenID Connect Provider federations

You can now configure support for an OpenID Connect Provider by using the API Protection user interface panel.

The API Protection panel provides user controls for enabling OIDC and specifying settings, such as issuer, point of contact, metadata URI, ID token encryption, and certificate usage. You can also specify Attribute Mappings for use in customizing claims. See Creating an API protection definition.

OpenID Connect Relying Party federations

Security Verify Access supports OpenID Connect Relying Party federations.

When configuring a OpenID Connect Relying Party two entities must be created - a federation and a partner. There can be multiple partners per federation, but each partner has only one federation.

The OpenID Connect Relying Party federation does not do anything on its own - it just serves as a container for the partners. Each OpenID Connect Relying Party Partner is a entity which consumes identities from a given OpenID Connect Provider. For more information on Relying Parties see:

- http://openid.net/specs/openid-connect-basic-1_0.html
- http://openid.net/specs/openid-connect-implicit-1_0.html

Support for Relying Party federations is enhanced in Security Verify Access Version 9.0.4. The enhancement include new configuration wizards. If you manage an existing (prior to Version 9.0.4) Relying Party federation, use the legacy wizards instead. Refer to the configuration instructions for your deployment:

- To deploy and manage new Relying Party federations, see "Authentication with OpenID Connect Relying Party" on page 84.

Authentication with OpenID Connect Relying Party

Security Verify Access supports authentication with OpenID Connect (OIDC) Relying Party.

An OIDC Relying Party is an OAuth 2.0 Client application that requires user authentication and claims from an OpenID Connect Provider. Security Verify Access supports Relying Party (RP) as part of the support of the OAuth 2.0 and OpenID Connect (OIDC) specifications.

Deployment of a Relying Party requires knowledge of OIDC concepts, work flows, and end points. For an overview of the Security Verify Access support for OIDC, see the following topics:

- OpenID Connect concepts
- OAuth 2.0 and OIDC workflows
- OAuth 2.0 endpoints

An OpenID Connect (OIDC) Relying Party (RP) is an OAuth client plus an identity management layer. You can invoke an RP connection to Security Verify Access to log a user into WebSEAL. The Security Verify Access implementation of the Relying Party for use during authentication includes, in addition to basic RP functions, an initiation delegate, a reentry delegate, and a context object. The Security Verify Access RP uses the following Security Verify Access features:

- Secure Token Service (STSUU)
- Verify Access credentials (iv-cred)
- JSON Web Token (JWT)
- Identity Mapping
- HTTP callout
- Attribute Mapping

The Security Verify Access RP supports the following OpenID Connect (OIDC) features:

- The OIDC Authorization code, OIDC implicit, and OIDC Hybrid flows.
- 256, 384, and 512-bit SHA signing algorithms for the types HS, RS, and ES.

  For a complete list, see "OpenID Connect Relying Party partner properties" on page 99.
- Use of `response_mode=form` post.
- The RP always sends a `state` and `nonce` for implicit flows.
- Encrypted ID tokens.
- Consumption of OIDC Provider (OP) metadata at run time, for easy configuration.

Relying party endpoints for authentication

A Secure Verify Access OpenID Connect Relying Party (RP) federation uses two URL endpoints.

Initiation or Kickoff URL

The user accesses this URL to initiate an OpenID Connect (OIDC) federated single sign-on. Access to this URL results in a redirect to the `/authorize` endpoint of the configured OIDC Provider.

```
https://<reverseproxy_host, port, junction>/sps/oidc/rp/<federation name>/kickoff/<partner name>
```

This endpoint supports providing a `Target` parameter. A `Target` query string parameter can be provided to define a location to redirect the user after a successful authentication. An example location is an application's landing page. This target must be a fully qualified URL containing protocol, host, and path information.

For example, with a junction of `/isam`, a federation of `my_federation` and a partner of `partner_company`, the URL is:

```
https://my.webseal.com/isam/sps/oidc/rp/my_federation/kickoff/partner_company
```

Reentry or Redirect URL

After the request to `/authorize` is made through the Kickoff URL, the user is redirected back to the RP through the Redirect URI. The URI value is included in the request to `/authorize`.

```
https://<reverseproxy host, port, junction>/sps/oidc/rp/<federation name>/redirect/<partner name>
```

The Redirect URL must be configured on the OIDC Provider. If this URL is accessed without a session state existing (that is, without first accessing the Kickoff URL), an error occurs. Assuming a successful flow, the user is authenticated after this URL is accessed.

For example, with a junction of `/isam`, a federation of `my_federation` and a partner of `partner_company`, the URL is:

```
https://my.webseal.com/isam/sps/oidc/rp/my_federation/redirect/partner_company
```

Relying party authentication flow

Relying Party authentication supports implicit flow and authorization code flow.

- Implicit flow is useful when you need to single sign-on from the internet to an intranet site. During the implicit flow, the `/token` endpoint is not needed, and no direct connection exists from the RP to the OpenID Provider (OP). This lack of direct connection is based on the assumption that no metadata or `/userinfo` is configured.
- Authorization code is traditionally considered to be more secure. It is the only instance where a refresh token might be issued. Both the ID token and access token are considered less at risk when the RP uses the authorization code flow, as neither is ever transported through the browser.

When a federated single sign-on is performed with the OpenID Connect (OIDC) Relying Party (RP), several steps must be completed. It is useful to understand these steps, so that you understand what potential customization can be made to the requested authentication.

1. The first step of an authentication is the kickoff, which is initiated by accessing the following URL:

   ```
   https://www.mywebseal.com/<isva junction>/sps/oidc/rp/<federation name>
   /kickoff/<partner name>
   ```

   For example, with a junction of `/isva`, a federation of `my_federation` and a partner of `partner_company,` the URL is:

   ```
   https://www.mywebseal.com/isva/sps/oidc/rp/my_federation/kickoff/partner_company
   ```

2. When the Kickoff request is first received, the federation and partner name are checked to ensure that the request is for federation with a valid configuration. The OIDC OP metadata, if configured, is retrieved now.

   See "Relying party authentication metadata" on page 87.
3. If a Target query parameter was provided, it is stored in the user's session.
4. The incoming HTTP request is then serialized into an `STSUniversalUser`(STSUU). This structure contains any incoming request parameters. Any parameters that must be added to the request to `/authorize` are added to STSUU context attributes.
5. If an advanced mapping rule is configured, it is run now. This action occurs now so that the authentication request to the OP can be modified at run time.

   See "Relying party advanced configuration" on page 88.
6. After the advanced mapping rule is invoked, the `state` and `response_type` are validated. When validated, they are persisted in the user session.

   See "Managing Distributed Session Cache" on page 281.
7. A response is then sent to the user agent, redirecting the user to the OIDC Provider (OP).

8. The OP completes its processing steps. Typically an authentication is performed, or a pre-existing session is checked, and potentially a prompt to consent is issued
9. When the OP processing completes, the user is redirected back to the RP, through the pre-registered redirect URI.

   This URI has the format:

   ```
   https://wwww.mywebseal.com/isvajct/sps/oidc/rp/<federationName>/redirect/<partnerName>
   ```

   For example, with the example values from above, the URL would be:

   ```
   https://wwww.mywebseal.com/isva/sps/oidc/rp/my_federation/redirect/partner_company
   ```

10. Next, the Redirect URI of the RP is invoked. This invocation occurs in one of the following ways:
    - If a GET request is serviced from the user agent, whether coming through a 302 from the OP or another method, and the `state` parameter is not included through a query string, then the RP sends back the self-posting form. The self-posting form extracts the fragment portion of the URL and posts the values to the RP.

      Note: The self-posting form is the template page `form_post.html`. Use the local management interface to obtain this file. Access **Federation** > **Global Settings** > **Template Files**. The path to the file is `C > oidc > rp > form_post.html`.
    - If the request is a POST, the incoming parameters are validated and the single sign-on proceeds.
    - 302 including query string. In this case, the RP processes the query parameters.

      Note: The OAuth RFC forbids this action when an access_token is included in the redirect.
11. Once the RP receives the redirect parameters from the OP, through one of the mechanisms above, the RP validates the request. The validation includes validating the state, and asserting that the incoming request contains all of the `response_types` included in the request.
12. Next, the advanced mapping rule is invoked. This invocation can be used to perform an HTTP Callout, or to add more parameters to the `/token` or `/userinfo` requests (if configured).
13. When the request is validated, if an id_token was returned from the request, it is validated and decrypted. The claims and header of this JWT are added to the STSUU. The `at_hash`, `nonce`, and `c_hash` claims of the id_token are validated.
14. After the implicit id_token is validated, if a code was returned it is now exchanged at the token endpoint. The response is then validated and the response parameters are added to the STSUU context attributes.
15. The id_token that was returned from the token endpoint is then validated and decrypted, and its claims are checked.
16. If the RP is configured to access user information, and possesses an access token, it makes a request to `/userinfo` with the access token. The response is included in the STSUU attribute list. The `sub` claim that is returned is checked against the `sub` claim in the id_token.
17. The STSUU is now passed to the final identity map step, where the STSUU is processed into a credential. This step is performed by HTTP callout or JavaScript mapping rule, depending on configuration.

    See "Relying Party identity mapping" on page 87.
18. If you're authenticating a user that does not exist in the Security Verify Access registry, the point of contact configuration must be updated to reflect this authentication.

    See Point of contact profiles for Federation.

Relying party authentication metadata

Metadata is the discovery information that the OpenID Provider (OP) exposes.

If metadata is configured, the Relying Party (RP) uses it as the source of the `/authorize`, `/token`, `/jwks`, and `/userinfo` URLs for the RP. The RP uses other metadata fields, including supported signing algorithms and supported response types. If the RP is configured for a particular `response_type` and signing algorithm, which are not included in the metadata, the metadata is still used. The OP advertises that it supports this action.

The RP overwrites any configured field with the metadata it retrieves. In some cases, the RP does not even prompt for configuration if it knows that metadata is provided. For example, it is impossible to provide an `/authorize` URL when you use metadata because metadata must expose `/authorize`.

In other instances, the RP permits you to elect which value to use, but ignores that value at run time if the value is incompatible with the metadata. For example, if you select ES256 as the signing algorithm, but the OP supports only RS256, the RP expects an RS256 signed JWT (not an ES256 signed JWT) because the OP advertises that it does not provide an ES algorithm. However, if the OP does support ES256, ES384 and ES512, and you select ES256, then this signature algorithm is used because it is an elected and compatible preference.

See the OpenID Connect specification for metadata definition: http://openid.net/specs/openid-connect-discovery-1_0.html#ProviderMetadata

Relying Party identity mapping

Identity mapping is a step in the Secure Verify Access federation flow that is invoked at the end of a successful single sign-on. Identity mapping can take place either through a JavaScript mapping rule or an HTTP callout.

Use of a JavaScript mapping allows administrators to modify the attributes of the session that was created for a user as a result of a federated single sign-on. Modification of attributes can be necessary because incoming data comes in several different forms, such as SAML assertions or a JSON Web Token (JWT). Sometimes this information is not complete, and more work is need to retrieve the entire profile. In other cases, the values are complete, but not in the correct form for a consuming application.

Note: For information on using HTTP callout, see External user mapping.

During the Relying Party flow, the credential (iv-cred) is built and returned to WebSEAL. At this step in the authentication process, you can use Relying Party identity mapping to perform the following actions, as needed.

- Set the principal name.

  The identity mapping step of a Relying Party must set a valid principal name to use in the user session. A common way to do this is to combine claims from a JWT, such as combining the `iss` and `sub` claims. See the example mapping rule below.
- Map attributes from the id_token or `/userinfo` into any additional credential attributes to be present in the users session.
- Access a protected resource by using the provided access_token (if one was issued), to retrieve more information to include in the session.
- Produce an attribute from the various claim sources, such as JWT claims, UserInfo, or additional callouts.
- Combine multiple attributes into a single more-consumable attribute.
- Make an advanced call to `/userinfo`, when the default callout that is provided with Security Verify Access is not sufficient.
- Persist access or refresh tokens.
- Use UserLookupHelper to perform just in time (JIT) provisioning of a Security Verify Access account.
- HTTP callouts

- Callout to APIs with the access_token.

JavaScript mapping rules call Java™ code from JavaScript. The set of classes that can be called are restricted. Examples include:

- `packages.com.ibm.security.access.user.UserLookupHelper`
- `com.ibm.security.access.httpclient.HttpClient`
- `com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils`

To view the full list of whitelisted (allowable) JavaScript classes, see "JavaScript whitelist" on page 265.

Identity mapping uses Secure Token Service (STS) chains. The chains use a Secure Token Service Universal User (STSUU) module to map the necessary attributes. For information on the Relying Party's use of the STSUU, see "Use of STSUU for the Relying Party" on page 91 and "Security Token Service Universal User document" on page 132.

When identity mapping includes a user identity that does not exist with the registry, the Security Verify Access point of contact must be configured accordingly.

- Review how to change the point of contact configurations for the Security Verify Access runtime, on the federation side. See "Point of contact profiles" on page 286.
- Understand how external authentication (EAI) works for users that exist within the registry. See External authentication interface HTTP header reference.

Example rule

The Security Verify Access distribution includes an example identity mapping rule for Relying Party. To view it:

1. Log in to the local management interface.
2. Select **Federation** > **Global Settings** > **Mapping Rules**.
3. Select `OIDCRP Category OIDC`, and use the **Edit** function to view the contents.

The example identity mapping rule `OIDCRP Category OIDC` demonstrates code that completes the mandatory requirement of an identity mapping rule to assign the Principal Name. The example assigns the name by combining the values of the `iss` and `sub` fields of the id_token. See the extract from the mapping rule below.

```
/*
 * Construct a basic identity made up of iss and sub
 */
var iss = stsuu.getAttributeContainer().getAttributeValueByName("iss");
var sub = stsuu.getAttributeContainer().getAttributeValueByName("sub");

/*
 * This builds a principal name from the iss and sub fields of the id_token. If
 * this user does not exist in the Verify Access registry, either modify to map to a
 * local user that is in the registry, or change the EAI authentication
 * settings of the federation runtime to use PAC authentication.
 */
stsuu.setPrincipalName(iss + "/" + sub)
```

To use PAC authentication, log in to the local management interface. Select **Federation** > **Point of Contact**. Select `Verify Access Credential` and click **Set as Current**.

Relying party advanced configuration

You can use advanced configuration to customize requests that are made by the Relying Party.

Advanced configuration consists of a JavaScript mapping rule, which you can configure on a per partner or per federation basis. The JavaScript mapping rule is invoked at the following points during the Relying Party (RP) single sign-on flow.

- Before the redirect to `/authorize`

- Before the request to `/token`

  This invocation point also allows the modification of the request to `/userinfo`.

The goal of the mapping rule is to add, augment, or remove parameters from the request that is about to be made. This request is achieved through a 302 redirect that is sent to the user agent.

Potential uses of the advanced mapping rule before the redirect from the OpenID Connect Provider (OP):

- Execution of an HTTP callout to a remote service.
- Modification of the request parameters for the request to `/authorize` that is about to be made. This modification can include changes to `request_type`, `redirect_uri`, `state`, `nonce`, `scope`, and any other parameters that the RP configuration includes.
- Addition of any new parameters. For example, if you are federating to Microsoft Azure, you might want to include a `resource` parameter. Common or standardized parameters include a `claims`, `prompt`, or `acr_values`.
- Removal of unwanted parameters, such as removing `response_mode=form_post`. For example, if the OP does not support `nonce` on authorization code flows, use this mechanism to remove it.

The RP uses the STSUniversalUser (STSUU) to represent the authentication request or response in its processing. The STSUU is passed into advanced mapping. HTTP parameters are included in the context attributes. JWT claims and the response from `/userinfo` are included in the attribute list, as they pertain to a users identity.

The processing flow before and after invocation of the rule is as follows:

1. An incoming single sign-on request is received, either as part of a single sign-on kickoff, or as a redirect from the OpenID Connect Provider (OP).
2. The request is unpacked into an STSUU structure.
3. The mapping rule adds, to the STSUU, any values that are configured in the RP that need to be sent as part of this authorize request. For example, `scope`, `client_id`, and `response_types`.
4. The advanced configuration rule is invoked, and changes are made to the STSUU.
5. When the rule is successfully run, the STSUU is converted into an HTTP request.

Supported scenarios:

- Augmentation of a single sign-on request at run time. This request is to `/authorize`.

  By unpacking the incoming single sign-on request at the `/kickoff` delegate, you can build an RP that tailors its functions based on the incoming request. Because the request parameters to `/kickoff` are included in the STSUU, you can modify the request to `/authorize`, based on the initial request to `/kickoff`. For example, a different scope might be requested, or a decision made against the use of an implicit flow. This mapping rule allows those values to be added, changed, or removed at run time.
- Allowance for OPs that do not fully conform to the OpenID Connect 1.0 specification.

  Some OPs might place extra requirements on incoming request parameters, or might support alternative parameter values. Some OPs might reject some of the specification-compliant values due to their own limitations.
- A similar scenario applies to requests to `/token`. A client might be required to present another parameter to `/token`, to use an additional feature that is supported or required by the OP.

Since the same rule runs at both points, authors of rules must include logic to extract the current request type, and run the logic only where appropriate. This requirement is similar to the OAuth concept `request_type`. Because of this requirement, the `operation` parameter in the STSUU can be used.

In your mapping rule, use the attribute `operation`, of type `urn:ibm:SAM:oidc:rp:operation` to execute the mapping rule code for the STSUU operation for the intended entry point. You can then specify attributes of the necessary attribute type, based on the operation value.

For more information, see:

- “Relying Party attribute types” on page 90

- "Use of STSUU for the Relying Party" on page 91
- "Security Token Service Universal User document" on page 132

Example rule

The Security Verify Access distribution includes an example advanced configuration mapping rule for Relying Party. This example demonstrates a simple way to include an example claims parameter. To view it:

1. Log in to the local management interface.
2. Select **Federation** > **Global Settings** > **Mapping Rules**.
3. Select `OIDCRP_ADV Category OIDC`, and use the **Edit** function to view the contents.

Relying Party attribute types

The Relying Party can use specific attribute types during advanced configuration.

In mapping rules, the attribute `operation`, of type `urn:ibm:SAM:oidc:rp:operation` runs the mapping rule code for the STSUU operation for the intended entry point. Based on the operation type, attributes of the necessary attribute type are specified, as shown in the following tables.

- If operation is `authorize`, add attributes before the redirect to `/authorize`.

Table 78. Attribute types to use before the redirect to `/authorize`

Attribute type	**Description**	**Usage**
urn:ibm:SAM:oidc:rp:kickoff:param	OIDC kickoff request parameters	read
urn:ibm:SAM:oidc:rp:authorize:uri	The authorization URL	read, write
urn:ibm:SAM:oidc:rp:authorize:req:param	OIDC `/authorize` request parameters. Can also be used to include a `claims` object in the request to `/authorize`. This claims parameter requests the email claim in the id_token as essential.	read, write

- If operation is `token`, add attributes before the request to `/token`. This operation means a request to `/redirect` for reentry from the OIDC Provider (OP).

Table 79. Attribute types to use before the request to `/token`, and to modify requests to `/userinfo`

Attribute type	**Description**	**Usage**
urn:ibm:SAM:oidc:rp:authorize:rsp:param	Use this type to retrieve any of the parameters that were included in the redirect from the OIDC Provider.	read
urn:ibm:SAM:oidc:rp:userinfo:req:param	Use this type to add more string query parameters to the request to `/userinfo`.	write
urn:ibm:SAM:oidc:rp:token:req:param	Use this type to add more body parameters to the `/token` request.	write
urn:ibm:SAM:oidc:rp:meta	Contains metadata values that are relevant to the entire Relying Party flow. For example, it includes the nonce that is presented to `/authorize`.	read

Attribute types that are used for responses from `/authorize`, `/token`, and `/userinfo`.

Table 80. Attribute types to use for responses

Attribute type	Description	Usage
urn:id_token:attribute:implicit	If an id_token is returned from /`authorize`, the id_token claims have this type.	read
urn:id_token:attribute:token	If an id_token is returned from /`token`, the id_token claims have this type.	read
urn:ibm:SAM:oidc:rp:userinfo:rsp:param	If a `/userinfo` request is made, the response properties have this type.	read
urn:ibm:SAM:oidc:rp:token:rsp:param	If a request to `/token` is made, the response parameters have this type. For example, `access_token`, `expires_in`, and `scope`.	read
urn:ibm:SAM:oidc:rp:authorize:rsp:paramf	The response parameters from /`authorize`. For example, `state`. If an implicit flow is run, an access_token or id_token might be present with this type.	read

Use of STSUU for the Relying Party

Relying Party identity mapping and advanced configuration can use Secure Token Service Universal User (STSUU) modules to obtain needed data. You can view some example usages, and a sample of a complete STSUU.

Example: How to get an authorize response from the request to /authorize (authorization code)

If you have the following STSUU variable as XML:

```
<stsuuser:ContextAttributes>
 ...
       <stsuuser:Attribute name="code" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
         <stsuuser:Value>d44df5efb1008969e26ce702ff0989e57448b809..8329</stsuuser:Value>
       </stsuuser:Attribute>
 ....
 </stsuuser:ContextAttributes>
```

Then you could use the following JavaScript code:

```
// For example, getting the authorization code. Take note that it is sourced from the context
 attributes.
var azn_code = stsuu.getContextAttributes().getAttributeValueByNameAndType("code",
  "urn:ibm:SAM:oidc:rp:authorize:rsp:param");
```

The code returns the value:

```
d44df5efb1008969e26ce702ff0989e57448b809..8329
```

Example: How to get a token response parameter (access token)

If you have the following STSUU variable as XML:

```
    <stsuuser:ContextAttributes>
     ...
           <stsuuser:Attribute name="access_token" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
              <stsuuser:Value>ya29.Gl39BHO35g7mBjJKkQNi0mS0rVeEvpxt9nLRfoOW0noKtvz4gUUiP3tz6-
TqJKgi62yXaHDs1NZV5DI
              </stsuuser:Value>
            </stsuuser:Attribute>
     ....
     </stsuuser:ContextAttributes>
```

Then you could use the following JavaScript code:

```
// For example, getting the authorization code. Take note that it is sourced from the context
 attributes.
var access_token = stsuu.getContextAttributes().getAttributeValueByNameAndType("access_token",
   "urn:ibm:SAM:oidc:rp:token:rsp:param");
```

The code returns the value:

```
ya29.Gl39BHO35g7mBjJKkQNi0mS0rVeEvpxt9nLRfoOW0noKtvz4gUUiP3tz6-TqJKgi62yXaHDs1NZV5DI
```

Example: How to get a parameter from the id_token from `/authorize`

If you have the following STSUU variable as XML:

```
<stsuuser:AttributeList>
...
    <stsuuser:Attribute name="email" type="urn:id_token:attribute:implicit">
      <stsuuser:Value>testuser@example.com</stsuuser:Value>
    </stsuuser:Attribute>
...
</stsuuser:AttributeList>
```

Then you could use the following JavaScript code:

```
// For example, getting the authorization code. Take note that its sourced from the context attributes.
// Take note that the attribute list is used, not the context attributes.
var email =
 stsuu.getAttributeContainer().getAttributeValueByNameAndType("email","urn:id_token:attribute:implicit");
```

The code returns the value:

```
testuser@example.com
```

Example: How to get a parameter from the id_token from `/token`

If you have the following STSUU variable as XML:

```
  <stsuuser:AttributeList>
...
    <stsuuser:Attribute name="email" type="urn:id_token:attribute:implicit">
      <stsuuser:Value>testuser2@example.com</stsuuser:Value>
    </stsuuser:Attribute>
...
  </stsuuser:AttributeList>
```

Then you could use the following JavaScript code:

```
// For example, getting the authorization code. Take note that its sourced from the context attributes.
// Take note that the attribute list is used, not the context attributes.
var email =
 stsuu.getAttributeContainer().getAttributeValueByNameAndType("email","urn:id_token:attribute:token");
```

The code returns the value:

```
testuser2@example.com
```

Example: How to get a parameter that came from `/userinfo`

If you have the following STSUU variable as XML:

```
  <stsuuser:AttributeList>
  ...
    <stsuuser:Attribute name="name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>Test User</stsuuser:Value>
    </stsuuser:Attribute>
  ...
  </stsuuser:AttributeList>
```

Then you could use the following JavaScript code:

```
// For example, getting the users 'name', this comes from the attribute list,
// as thats where userinfo response parameters go.
var name =
 stsuu.getAttributeContainer().getAttributeValueByNameAndType("name","urn:ibm:SAM:oidc:rp:userinfo:rsp:param");
```

The code returns the value:

```
Test User
```

A complete STSUU of an authorization code

```
<?xml version="1.0" encoding="UTF-8"?>
<stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
  <stsuuser:Principal/>
  <stsuuser:AttributeList>
    <stsuuser:Attribute name="family_name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>User</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="email" type="urn:id_token:attribute:token">
      <stsuuser:Value>testuser@example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="family_name" type="urn:id_token:attribute:token">
      <stsuuser:Value>User</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="email_verified" type="urn:id_token:attribute:token">
      <stsuuser:Value>true</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="exp" type="urn:id_token:attribute:token">
      <stsuuser:Value>1510105195</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="name" type="urn:id_token:attribute:token">
      <stsuuser:Value>Test User</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="email_verified" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>true</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="picture" type="urn:id_token:attribute:token">
      <stsuuser:Value>https://lh6.example.com/-xfh8mrdMtRk/AAAAAAAAAAI/AAAAAAAAAAA/
                 ANQ0kf7mUOsYQEP0mNtQgWDQRrSy9hvVnA/s96-c/photo.jpg</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="aud" type="urn:id_token:attribute:token">
      <stsuuser:Value>269072228812-th7t9u11fnk6but52c7u6rfhkqrkldha.example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>Test User</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="email" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>testuser@example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="iat" type="urn:id_token:attribute:token">
      <stsuuser:Value>1510101595</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="given_name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>Test</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="locale" type="urn:id_token:attribute:token">
      <stsuuser:Value>en</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="given_name" type="urn:id_token:attribute:token">
      <stsuuser:Value>Test</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="iss" type="urn:id_token:attribute:token">
      <stsuuser:Value>https://example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="sub" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>111172479139097978803</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="at_hash" type="urn:id_token:attribute:token">
      <stsuuser:Value>4kiED05hW5JX45rFxFAqmQ</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="azp" type="urn:id_token:attribute:token">
      <stsuuser:Value>269072228812-th7t9u11fnk6but52c7u6rfhkqrkldha.example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="locale" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>en</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="picture" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
      <stsuuser:Value>https://lh3.example.com/-XdUIqdMkCWA/AAAAAAAAAAI/AAAAAAAAAAA/4252rscbv5M/photo.jpg
      </stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="sub" type="urn:id_token:attribute:token">
      <stsuuser:Value>111172479139097978803</stsuuser:Value>
    </stsuuser:Attribute>
  </stsuuser:AttributeList>
  <stsuuser:RequestSecurityToken>
    <stsuuser:Attribute name="Issuer" type="http://schemas.xmlsoap.org/ws/2005/02/trust">
      <stsuuser:Value>https://accounts.example.com</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="AppliesTo" type="http://schemas.xmlsoap.org/ws/2004/09/policy">
      <stsuuser:Value>https://www.mysp.mycompany.com/goog/sps/oidc/rp/test:ivc:metaRP</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="Forwardable" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>true</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="RenewingOk" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>false</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="RenewingAllow" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>true</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="AllowPostDating" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>false</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="KeySize" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>0</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="RequestType" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="Base" type="urn:ibm:names:ITFIM:1.0:stsuuser">
      <stsuuser:Value>
```

```
         <stsuuser:STSUniversalUser>
           <stsuuser:Principal/>
           <stsuuser:AttributeList>
             <stsuuser:Attribute name="family_name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>User</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="email" type="urn:id_token:attribute:token">
               <stsuuser:Value>testuser@examplecom</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="family_name" type="urn:id_token:attribute:token">
               <stsuuser:Value>User</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="email_verified" type="urn:id_token:attribute:token">
               <stsuuser:Value>true</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="exp" type="urn:id_token:attribute:token">
               <stsuuser:Value>1510105195</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="name" type="urn:id_token:attribute:token">
               <stsuuser:Value>Test User</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="email_verified" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>true</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="picture" type="urn:id_token:attribute:token">
               <stsuuser:Value>https://lh6.example.com/-xfh8mrdMtRk/AAAAAAAAAAI/AAAAAAAAAAA/
                        ANQ0kf7mUOsYQEP0mNtQgWDQRrSy9hvVnA/s96-c/photo.jpg</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="aud" type="urn:id_token:attribute:token">
               <stsuuser:Value>269072228812-th7t9u11fnk6but52c7u6rfhkqrkldha.example.com</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>Test User</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="email" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>testuser@example.com</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="iat" type="urn:id_token:attribute:token">
               <stsuuser:Value>1510101595</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="given_name" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>Test</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="locale" type="urn:id_token:attribute:token">
               <stsuuser:Value>en</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="given_name" type="urn:id_token:attribute:token">
               <stsuuser:Value>Test</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="iss" type="urn:id_token:attribute:token">
               <stsuuser:Value>https://accounts.example.com</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="sub" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>111172479139097978803</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="at_hash" type="urn:id_token:attribute:token">
               <stsuuser:Value>4kiED05hW5JX45rFxFAqmQ</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="azp" type="urn:id_token:attribute:token">
               <stsuuser:Value>269072228812-th7t9u11fnk6but52c7u6rfhkqrkldha.example.com</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="locale" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>en</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="picture" type="urn:ibm:SAM:oidc:rp:userinfo:rsp:param">
               <stsuuser:Value>https://lh3.example.com/-XdUIqdMkCWA/AAAAAAAAAAI/AAAAAAAAAAA/4252rscbv5M/
photo.jpg
               </stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="sub" type="urn:id_token:attribute:token">
               <stsuuser:Value>111172479139097978803</stsuuser:Value>
             </stsuuser:Attribute>
           </stsuuser:AttributeList>
           <stsuuser:RequestSecurityToken/>
           <stsuuser:ContextAttributes>
             <stsuuser:Attribute name="prompt" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
               <stsuuser:Value>none</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="authuser" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
               <stsuuser:Value>0</stsuuser:Value>
             </stsuuser:Attribute>
             <stsuuser:Attribute name="id_token" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
```

```
 <stsuuser:Value>eyJhbGciOiJSUzI1NiIsImtpZCI6ImExMmMzYTYxMDkxOTMzMDgzMDA3OTkyNWRmOWM5NWFkODUyNzQ1ODAifQ.

 eyJhenAiOiIyNjkwNzIyMjg4MTItdGg3dDl1MTFmbms2YnV0NTJjN3U2cmZoa3Fya2xkaGEuYXBwcy5nb29nbGV1c2VyY29udGVudC5

 jb20iLCJhdWQiOiIyNjkwNzIyMjg4MTItdGg3dDl1MTFmbms2YnV0NTJjN3U2cmZoa3Fya2xkaGEuYXBwcy5nb29nbGV1c2VyY29udGV

 udC5jb20iLCJzdWIiOiIxMTExNzI0NzkxMzkwOTc5Nzg4MDMiLCJlbWFpbCI6ImxtZi5vaWRjLnRlc3RAZ21haWwuY29tIiwiZW1haWx

 fdmVyaWZpZWQiOnRydWUsImF0X2hhc2giOiI0a2lFRDA1aFc1Slg0NXJGeEZBcW1RIiwiaXNzIjoiaHR0cHM6Ly9hY2NvdW50cy5nb29

 nbGUuY29tIiwiaWF0IjoxNTEwMTAxNTk1LCJleHAiOjE1MTAxMDUxOTUsIm5hbWUiOiJUZXN0IFVzZXIiLCJwaWN0dXJlIjoiaHR0cHM

 6Ly9saDYuZ29vZ2xldXNlcmNvbnRlbnQuY29tLy14Zmg4bXJkTXRSay9BQUFBQUFBQUFBSS9BQUFBQUFBQUFBQS9BTlEwa2Y3bVVPc1l

 RRVAwbU50UWdXRFFSclN5OWh2Vm5BL3M5Ni1jL3Bob3RvLmpwZyIsImdpdmVuX25hbWUiOiJUZXN0IiwiZmFtaWx5X25hbWUiOiJVc2V
              yIiwibG9jYWxlIjoiZW4ifQ.L-tUdSUTHwkmj6VjOFgoGXnAnFEGe179x1ZiIReWc6t6rN7RvQrTlIxLhs3z_P-Ec-
fAQg1UGwXsU545
              Z4TNkif4UDT2JkDPIxaY746oAGZyKZcUm7Lxw6n1tOzp3c8tYRaVty-
R8840rI1ALUExOYv72BRlTyQG7o7FZjs_D1lMnGvPe6fwzPmT
              -
ShjhjYu2joZmsJ07uPUFPLBWDhMwN7hUcPnbqWQpypJmKN7EQBKpJImz8vMkAVVxNSJpeU09dNICLh5MkNEsoIcKCDsYK4o1N_SaRLYh
              xYIhUgbT_-l4f5fvRv5W1AQwn-v4L220gF9vYrb1rYktvMV9fFYGQ</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="token_type" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
              <stsuuser:Value>Bearer</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="signature" type="urn:id_token:attribute:token">
              <stsuuser:Value>L-tUdSUTHwkmj6VjOFgoGXnAnFEGe179x1ZiIReWc6t6rN7RvQrTlIxLhs3z_P-Ec-
fAQg1UGwXsU545Z4TNkif
              4UDT2JkDPIxaY746oAGZyKZcUm7Lxw6n1tOzp3c8tYRaVty-
R8840rI1ALUExOYv72BRlTyQG7o7FZjs_D1lMnGvPe6fwzPmT-ShjhjY

 u2joZmsJ07uPUFPLBWDhMwN7hUcPnbqWQpypJmKN7EQBKpJImz8vMkAVVxNSJpeU09dNICLh5MkNEsoIcKCDsYK4o1N_SaRLYhxYIhUgb
              T_-l4f5fvRv5W1AQwn-v4L220gF9vYrb1rYktvMV9fFYGQ</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="session_state" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
              <stsuuser:Value>d44df5efb1008969e26ce702ff0989e57448b809..8329</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="expires_in" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
              <stsuuser:Value>3600</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="response_type" type="urn:ibm:SAM:oidc:rp:meta">
              <stsuuser:Value>code</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="nonce" type="urn:ibm:SAM:oidc:rp:meta"/>
            <stsuuser:Attribute name="access_token" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
              <stsuuser:Value>ya29.Gl39BHO35g7mBjJKkQNi0rM2CSeW_x0GF_LgWdX0udmIa0HzD-yyzfKikfRYWU_JK_E-im
              S0rVeEvpxt9nLRfoOW0noKtvz4gUUiP3tz6-TqJKgi62yXaHDs1NZV5DI</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="header" type="urn:id_token:attribute:token">
              <stsuuser:Value>{"alg":"RS256","kid":"a12c3a61091933083007992 5df9c95ad85274580"}</
stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="state" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
              <stsuuser:Value>8lj0Nv0Wzm</stsuuser:Value>
            </stsuuser:Attribute>
            <stsuuser:Attribute name="code" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
              <stsuuser:Value>4/t5uPKRAieP6r9AbclAhzwK6gLUC8vmuULWDm1viYmMg</stsuuser:Value>
            </stsuuser:Attribute>
          </stsuuser:ContextAttributes>
          <stsuuser:AdditionalAttributeStatement/>
        </stsuuser:STSUniversalUser>
      </stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="Delegatable" type="com:tivoli:am:fim:sts:RST">
      <stsuuser:Value>false</stsuuser:Value>
    </stsuuser:Attribute>
  </stsuuser:RequestSecurityToken>
  <stsuuser:ContextAttributes>
    <stsuuser:Attribute name="prompt" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
      <stsuuser:Value>none</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="authuser" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
      <stsuuser:Value>0</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="id_token" type="urn:ibm:SAM:oidc:rp:token:rsp:param">

 <stsuuser:Value>eyJhbGciOiJSUzI1NiIsImtpZCI6ImExMmMzYTYxMDkxOTMzMDgzMDA3OTkyNWRmOWM5NWFkODUyNzQ1ODAifQ.

 eyJhenAiOiIyNjkwNzIyMjg4MTItdGg3dDl1MTFmbms2YnV0NTJjN3U2cmZoa3Fya2xkaGEuYXBwcy5nb29nbGV1c2VyY29udGVudC5
 jb20iLCJhdWQiOiIyNjkwNzIyMjg4MTItdGg3dDl1MTFmbms2YnV0NTJjN3U2cmZoa3Fya2xkaGEuYXBwcy5nb29nbGV1c2VyY29udGV

 udC5jb20iLCJzdWIiOiIxMTExNzI0NzkxMzkwOTc5Nzg4MDMiLCJlbWFpbCI6ImxtZi5vaWRjLnRlc3RAZ21haWwuY29tIiwiZW1haWx

 fdmVyaWZpZWQiOnRydWUsImF0X2hhc2giOiI0a2lFRDA1aFc1Slg0NXJGeEZBcW1RIiwiaXNzIjoiaHR0cHM6Ly9hY2NvdW50cy5nb29n

 bGUuY29tIiwiaWF0IjoxNTEwMTAxNTk1LCJleHAiOjE1MTAxMDUxOTUsIm5hbWUiOiJUZXN0IFVzZXIiLCJwaWN0dXJlIjoiaHR0cHM6L

 y9saDYuZ29vZ2xldXNlcmNvbnRlbnQuY29tLy14Zmg4bXJkTXRSay9BQUFBQUFBQUFBSS9BQUFBQUFBQUFBQS9BTlEwa2Y3bVVPc1lRRV

 AwbU50UWdXRFFSclN5OWh2Vm5BL3M5Ni1jL3Bob3RvLmpwZyIsImdpdmVuX25hbWUiOiJUZXN0IiwiZmFtaWx5X25hbWUiOiJVc2VyIiwi
       bG9jYWxlIjoiZW4ifQ.L-tUdSUTHwkmj6VjOFgoGXnAnFEGe179x1ZiIReWc6t6rN7RvQrTlIxLhs3z_P-Ec-
fAQg1UGwXsU545Z4TNkif
       4UDT2JkDPIxaY746oAGZyKZcUm7Lxw6n1tOzp3c8tYRaVty-R8840rI1ALUExOYv72BRlTyQG7o7FZjs_D1lMnGvPe6fwzPmT-
ShjhjYu2

 joZmsJ07uPUFPLBWDhMwN7hUcPnbqWQpypJmKN7EQBKpJImz8vMkAVVxNSJpeU09dNICLh5MkNEsoIcKCDsYK4o1N_SaRLYhxYIhUgbT_-
       l4f5fvRv5W1AQwn-v4L220gF9vYrb1rYktvMV9fFYGQ</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="token_type" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
      <stsuuser:Value>Bearer</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="session_state" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
      <stsuuser:Value>d44df5efb1008969e26ce702ff0989e57448b809..8329</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="expires_in" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
      <stsuuser:Value>3600</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="response_type" type="urn:ibm:SAM:oidc:rp:meta">
      <stsuuser:Value>code</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="nonce" type="urn:ibm:SAM:oidc:rp:meta"/>
    <stsuuser:Attribute name="access_token" type="urn:ibm:SAM:oidc:rp:token:rsp:param">
      <stsuuser:Value>ya29.Gl39BHO35g7mBjJKkQNi0rM2CSeW_x0GF_LgWdX0udmIa0HzD-yyzfKikfRYWU_JK_E-
imS0rVeEvpxt9nLRfoOW0
      noKtvz4gUUiP3tz6-TqJKgi62yXaHDs1NZV5DI</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="header" type="urn:id_token:attribute:token">
      <stsuuser:Value>{"alg":"RS256","kid":"a12c3a610919330830079925df9c95ad85274580"}</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="state" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
      <stsuuser:Value>8lj0Nv0Wzm</stsuuser:Value>
    </stsuuser:Attribute>
    <stsuuser:Attribute name="code" type="urn:ibm:SAM:oidc:rp:authorize:rsp:param">
      <stsuuser:Value>4/t5uPKRAieP6r9AbclAhzwK6gLUC8vmuULWDm1viYmMg</stsuuser:Value>
    </stsuuser:Attribute>
  </stsuuser:ContextAttributes>
  <stsuuser:AdditionalAttributeStatement id=""/>
</stsuuser:STSUniversalUser>
```

Configuring an OpenID Connect Relying Party federation

You can use the Federation page, in the local management interface, to configure an OpenID Connect Relying Party federation.

Procedure

1. Log in to the local management console.
2. Select **Federation** > **Manage** > **Federations**.
3. Click **Add**.
4. On the Federation Protocol page, enter a **Federation Name** and select **OpenID Connect Relying Party**.

   Note: Do not select **Legacy OpenID Connect (Provider or Relying Party)**. This selection is used only for maintaining existing legacy deployments of OpenID Connect federations. For information on configuring legacy federations, see Configuring a legacy relying party federation.
5. Supply values for the configuration properties as prompted on each page by the configuration wizard.

   For information on properties, see "OpenID Connect Relying Party federation properties" on page 98.
6. When you have completed the wizard pages, review the **Summary** page, and click **OK**.

7. Deploy the pending changes.

   Note: The deploy operation triggers a runtime restart.

OpenID Connect Relying Party federation properties

Define these properties when you configure an OpenID Connect Relying Party federation

Point of Contact

String containing the protocol, host, port and path of the runtime junction on the Reverse Proxy instance. This is used to automatically generate redirect URIs derived from the `applies to` value of the partner. An example value for this property is `https://www.reverse.proxy.com:443/mga`, where `www.reverse.proxy.com` is the hostname of the Reverse Proxy instance, 443 is the listening SSL port of the instance, and `/mga` is the local junction to the Federation runtime.

Default Response Types

An array of elements that specify the default flow type to run when metadata URL is specified. The flow types are authorization code, implicit flow, or any hybrid flow.

- code
- id_token
- token

For information on the use of response types in each flow, see OAuth 2.0 and OIDC workflows.

Attribute Mapping

You can use the Attribute Mapping page to define new attributes that can be used to customize claims from attribute sources. Attribute sources can be: Fixed, Credential, or LDAP.

To create a new mapping, select **New** and enter **Attribute Name**. Select **Attribute Source** type.

To remove an existing **Attribute Name**, select the attribute and click **Delete**.

Identity mapping

Identity mapping options

- **Do not perform identity mapping**
- **Use JavaScript transformation for identity mapping**
- **Use an external web service for identity mapping**

If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account.

If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.

If you choose JavaScript for mapping, on a subsequent page, you are asked to select the JavaScript file to use.

If you choose an external web service, on a subsequent page, you are asked to provide the following information:

- URI format (HTTP or HTTPS)
- Web service URI
- Server Certificate database, if the URI format is HTTPS.
- Client authentication type, if the URI format is HTTPS.
- Message format:
  - XML
  - WS-Trust

Advanced Configuration

Supported options:

- **Advanced configuration is not required**
- **Use JavaScript for advanced configuration**

You can use JavaScript to create mapping rules that add optional parameters to OpenID Connect requests. Open ID Connect requests can contain optional request parameters, as supported by the OIDC Provider. For example, `max_age`, `acr_values`, and claims.

If you choose to use JavaScript, the federation wizard displays existing advanced configuration mapping rules. Select the existing (already defined) JavaScript mapping rule that contains the advanced configuration that you want to use.

Configuring an OpenID Connect Relying Party partner

You can use the Partners action on the Federations page in the local management interface to configure an OpenID Connect Relying Party partner.

Procedure

1. Log in to the local management console.
2. Select **Federation** > **Manage** > **Federations**.
3. The existing federations are displayed in a list. Select the OpenID Connect Relying Party federation that you want to add a partner to.
4. Click **Partners**. Click **Add**.
5. Enter a name for the partner, and select the **Enabled** check box.

   The OIDC10 Connection Template is selected. The field is read-only.
6. Follow the UI wizard prompts to supply the required properties.

   For more information about what each field means, see "OpenID Connect Relying Party partner properties" on page 99.
7. When you have completed the wizard pages, review the **Summary** page, and click **OK**.
8. Deploy the pending changes.

   Note: The deploy operation triggers a runtime restart.

OpenID Connect Relying Party partner properties

Define these properties when you configure an OpenID Connect Relying Party partner.

Client ID
: Value that is used to identify this Relying Party at the OpenID Connect (OIDC) Provider. This value is required.

Client Secret
: Value that is used in combination with the Relying Party to authenticate at the OIDC Provider. Not specifying a Client Secret indicates that the client is public. Required to perform the Authorization Code grant, and to complete signing.

Metadata Endpoint
: The `/metadata` endpoint URL of the OIDC Provider.

Issuer Identifier

The expected value of the `iss` claim in a JWT. If this value does not match the contents of the JWT, then the authentication is rejected.

Response Types

An array of elements that specify the flow type to run when metadata URL is specified. The flow types are authorization code, implicit flow, or any hybrid flow.

- code
- id_token
- token

For more information, see OAuth 2.0 and OIDC workflows.

Authorization endpoint URL

The `/authorization` endpoint that is used to start the OpenID Connect flow at the OIDC Provider.

Token endpoint URL

The `/token` endpoint that is used to exchange an authorization code for an ID token and access token. Required if `code` response type is selected. Required to perform the Authorization Code grant. Requires a client secret to be set.

Signature Algorithm

Specifies the algorithm that is used to validate the JWT. See the next table for a list of valid values.

Table 81. Supported signature algorithms.

Digital Signature or MAC Algorithm	JWS `alg` parameter value
HMAC using SHA-2	HS256, HS384, HS512. Performs symmetric signing with the use of a client secret. A client secret is required.
RSASSA-PKCS1-V1_5 Digital Signatures with SHA-2	RS256, RS384, RS512. Performs asymmetric signing with the use of certificates. A JWK endpoint URL or a Signing Key keystore and label is required to perform RS256, RS384, and RS512 signing. RS256 is the default algorithm.
Elliptic Curve Digital Signatures (ECDSA) with SHA-2	ES256, ES384, ES512. Requires certificate.
A value of none denotes that no signing is performed on the issued JWT.	none

Signature validation behavior is determined by whether the Relying Party (RP) partner uses the OpenID Provider metadata.

- If the RP partner uses the OpenID Provider's metadata, and the metadata publishes more than one supported signing algorithm, then the RP uses its partner configuration to validate the signature.
- If the RP partner uses the OpenID Provider's metadata, and the metadata publishes only one supported signing algorithm, then the RP uses that single signing algorithm (as published by OpenID Provider's metadata) to validate the signature.
- If the RP partner does not use the OpenID Provider's metadata, then the RP use its partner configuration to validate the signature.

For more information, see https://bitbucket.org/b_c/jose4j/wiki/Home.

Use checked-in certificate

Select this check box on the **JWT Signature Verification** page if you want to use a certificate from an existing keystore for signing. If you select this option, you must select a keystore from the **Certificate Database** menu, and select a certificate from the **Certificate Label** field.

If you select this option, you cannot select the **JWK Endpoint URL** option.

Use JWK endpoint

Select this check box on the **JWT Signature Verification** page if you want to use the JWK endpoint of the OIDC provider. If you select this check box, you do not need to specify a Verification Certificate (**Certificate Database** and **Certificate Label**).

Certificate Database

When the signature algorithm requires a certificate, this property is the keystore that contains the selected certificate to perform the signing. When the signature algorithm does not require a certificate, this property is invalid. You cannot specify a **Certificate Database** when you specify a **Use JWK Endpoint**.

Certificate Label

When the signature algorithm requires a certificate, this property is the alias of the public key in the selected keystore (certificate database) to use in signature verification. You cannot specify a **Certificate Label** when you specify a **JWK Endpoint URL**.

JWK Endpoint URL

When the signature algorithm requires a certificate, this property is the JWK Endpoint of the OIDC provider. However, if the metadata endpoint is specified, the JWK URL can be read from metadata information.

This field is required if you do not specify a **Use checked-in certificate** and you specify an algorithm that requires JWT signatures.

Key Management Algorithm

The key management algorithm to use for JWT Decryption. The next table lists the supported algorithms.

Table 82. Key management algorithms.

Key Management Algorithm	**JWE `alg` parameter value**
The default value	none
Direct encryption with a shared symmetric key	`dir`
AES key wrap	A128KW, A192KW, and A256KW
AES GCM key encryption	A128GCMKW, A192GCMKW, and A256GCMKW
Elliptic Curve Diffie-Hellman Ephemeral Static key agreement using Concat KDF	ECDH-ES
Elliptic Curve Diffie-Hellman Ephemeral Static key agreement using Concat KDF with AES key wrap	ECDH-ES+A128KW, ECDH-ES+A192KW, ECDH-ES+A256KW
RSAES-PKCS1-V1_5 key encryption	RSA1_5
RSAES using OAEP key encryption	RSA-OAEP and RSA-OAEP-256

- When the selected algorithm requires a certificate, such as RSA or ECDH algorithms, both the **Certificate Database** and **Certificate Label** for the **Decryption Certificate** must be specified.
- For more information, see https://bitbucket.org/b_c/jose4j/wiki/Home.

Content Encryption Algorithm

The content encryption algorithm to use. The next table lists the supported algorithms.

Table 83. Content encryption algorithms.

Content Encryption Algorithm	**JWE "enc" Parameter Value**
The default value.	none

Table 83. Content encryption algorithms. (continued)

Content Encryption Algorithm	JWE "enc" Parameter Value
Authenticated encryption with AES-CBC and HMAC-SHA2	A128CBC-HS256, A192CBC-HS384, A256CBC-HS512
Authenticated encryption with Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM)	A128GCM, A192GCM, A256GCM

- If the key management algorithm is set to a value other than none, the content encryption algorithm must also be a value other than none.
- For more information, see https://bitbucket.org/b_c/jose4j/wiki/Home.

Decryption Certificate - Certificate Database

When the key management algorithm requires a certificate, this property is the certificate database (keystore) which contains the selected certificate to perform JWT decryption. When the key management algorithm does not require a certificate, this property is invalid.

Decryption Certificate - Certificate Label

When the key management algorithm requires a certificate, this property is the alias of the private key in the selected keystore to perform JWT decryption.

Scope

An array of strings that identify the scopes to request from the provider. Must contain `openid`. This property is an array of elements.

The default string is `openid`.

Userinfo Request - Perform userinfo request automatically

Boolean setting. Select this check box to specify whether to perform a UserInfo request automatically whenever possible.

Select this option if you want to populate the credential (`iv-cred`) from both the ID token and UserInfo. However, the `/userinfo` endpoint is optional for OIDC Providers. If your provider does not support the UserInfo endpoint, Security Verify Access cannot complete the request.

Keep in mind that a goal of Relying Parties is to retrieve user information, such as `given_name`, `family_name`, and`birthdate`, and then populate the credential. The user information is obtained from the ID token and – if the OIDC Provider supports the `/userinfo` endpoint – from the UserInfo response. The information that is returned in an ID token can differ from the information in `/userinfo`.

You can choose to populate the credential solely from the ID token that is returned during the selected flow. However, some flows do not have an ID Token, such as `response_type=token`. (The response_type can be any combination of `code`, `token`, and `id_token`). Choose whether to perform userinfo request automatically depending on whether your deployment provides `/userinfo`.

Token Endpoint Authentication Method

The token endpoint authentication method. Valid values:

- `client_secret_basic`
- `client_secret_post`

Attribute Mapping

You can use the **Attribute Mapping** page to define new attributes that can be used to customize claims from attribute sources. Attribute sources can be `Fixed`, `Credential`, or `LDAP`.

To create a new mapping, select **New** and enter **Attribute Name**. Select **Attribute Source** type.

To remove an existing **Attribute Name**, select the attribute and click **Delete**.

Identity mapping

Identity mapping options

- **Use the identity mapping that is configured for this partner's federation**
- **Do not perform identity mapping**
- **Use JavaScript transformation for identity mapping**
- **Use an external web service for identity mapping**

If you configure an identity provider, this mapping specifies how to create an assertion that contains attributes that are mapped from a local user account.

If you configure a service provider, this mapping specifies how to match an assertion from the partner to the local user accounts.

If you choose JavaScript for mapping, on a subsequent page, you are asked to select the JavaScript file to use.

If you choose an external web service, on a subsequent page, you are asked to provide the following information:

- URI format (HTTP or HTTPS)
- Web service URI
- Server Certificate database, if the URI format is HTTPS.
- Client authentication type, if the URI format is HTTPS.
- Message format:
  - XML
  - WS-Trust

Advanced Configuration

Use this configuration to customize the request. Supported options:

- **Use the advanced configuration that is configured for this partner's federation**
- **Advanced configuration is not required.**
- **Use JavaScript for advanced configuration**

You can use JavaScript to create mapping rules that add optional parameters to OpenID Connect requests. Open ID Connect requests can contain optional request parameters, as supported by the OIDC Provider. For example, `max_age`, `acr_values`, and claims.

If you choose to use JavaScript, the federation wizard displays existing advanced configuration mapping rules. Select the existing (already defined) JavaScript mapping rule that contains the advanced configuration that you want to use.

Making a request to `/userinfo` as part of authentication

You can add request parameters to the `/userinfo` request by using an advanced mapping rule.

You can configure the relying party to make a request to `/userinfo` as part of the authentication request. This request is useful when the ID Token does not contain complete identity information. Identity mapping is also needed to produce a valid subject in cases where an ID Token is not issued, and only an access token is available.

The configured `/userinfo` URL is invoked with the `Authorization: Bearer` header as defined by section 5.3.1 in the specification: http://openid.net/specs/openid-connect-core-1_0.html#UserInfoRequest

If metadata is configured, and no `/userinfo` URL is present in the metadata, then the `/userinfo` request is not made.

The successful `/userinfo` response is added to the STSUU attribute list. The attributes have the type `urn:ibm:SAM:oidc:rp:userinfo:rsp:param`.

You can add request parameters to the `/userinfo` request by using an advanced mapping rule.

Add context attributes with the type `urn:ibm:SAM:oidc:rp:userinfo:req:param` to include them in the query string of the request.

For example, this code adds a nonce value to the `/userinfo` request.

```
var nonce  = new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("nonce",
       "urn:ibm:SAM:oidc:rp:userinfo:req:param", "myNonce");
       stsuu.addContextAttribute(nonce);
```

After you create a mapping rule, you can add it to a Relying Party configuration. Use the Advanced Configuration page in the UI wizard, when either creating or editing a Relying Party federation or partner.

Conformance

IBM Security Verify Access supports FAPI conformance and OpenID Connect Discovery conformance.

Setting up the OIDC Definition API

Before you begin

To configure an API protection definition to be OIDC OP conformant and Financial Grade API compliant, ensure the OIDC Compliant and FAPI Compliant flag are checked. See OIDC Definition and WebSEAL OAuth Config.

Follow the guidelines below and the configuration steps in this topic to be completely conformed:

For both FAPI and OIDC
Ensure that the OIDC well-known endpoint is configured. See OpenID Connect Discovery.

For FAPI only

- Ensure that each client has a certificate and the public portion of that certificate is added to `rt_profile` or `signing ssl db` (required for Request JWT validation). The same client certificate can be added to `pdsrv` or `webseal ssl db` for MTLS. Ensure the certificate that is used for JWT validation is ES256 to meet FAPI requirements. See Configuring FAPI Client.
- FAPI requires the signing algorithm used for signing JWT to be ES256. Ensure a certificate where the algorithm that is mentioned is used, to be FAPI Compliant.
- Update Discovery Endpoint. The following parameters are required to be added to `metadata.json`.

  ```
  "claims_supported":
  ["realmName","preferred_username","given_name","uid","upn","groupIds","employee_id","name","tenantId","mobile_number",
  "tls_client_certificate_bound_access_tokens":<%var supported =
   true;templateContext.response.body.write(supported);%>
  ```

- Set `[session]` variable ‘`require-map`’ to '`yes`' in `webseal`. This ensures that HTTP headers are not valid session keys or authentication tokens unless they are received through an MPA. In FAPI, this functionality can be used to ensure each token and the certificate information are build as one unique session without any form of session caching.
- Set Point of Contact to **Access Manager Credential**.

About this task

Note: OIDC Compliance is a prerequisite for FAPI Compliance. The following conformances are configured when the OIDC or FAPI Wizards are checked.

More information on the functionalities that are performed can be found in OpenID Connect Provider Conformance and FAPI Conformance.

The following are configured when OIDC Compliant flag is check in API Definition API.

OIDC Conformance (OIDC definition)

- Access Policy – max_age and prompt=none
- Mapping Rule - authenticationTime
- Mapping Rule – produce_userinfo_jwt
- Mapping Rule – redirect_uri
- Mapping Rule – nonce
- Mapping Rule – assert_no_code_reuse
- STS Chain – Userinfo as JWT
- STS Chain – Request JWT (With a module for mapping rule and validate Request Object added by default this code only runs if FAPI flag is turned on in the definition)
- STS Chain – Client Authentication

FAPI Conformance

The following articles are configured when FAPI Compliant flag is checked in WebSEAL OAuth and OpenID Connect Provider Configuration and API Protected Definition accordingly.

WebSEAL - OAuth and OpenID Connect Provider Configuration (FAPI Compliant flag)	**OpenID Connect and API Protection (FAPI Compliant flag)**
Authentication Mechanism – FAPI Cert Authentication with FAPI_CertEAI.js (Available by default in Verify Access 10)	Mapping Rule – s_hash
WebSEAL Config – Configure FAPI Cert EAI	Mapping Rule – Disallow `response_type` code
WebSEAL Config – Configure HTTP Transformation for Sample Resource Endpoint	Mapping Rule – Disallow state in request parameter
Access Policy – isam_oauth_unauth acl to junction/sps/auth	Mapping Rule – Disallow state in request parameter
	STS Chain – Request JWT (With a module for mapping rule that triggers `FAPI_ValidateJWT.js`. This code only runs if FAPI flag is turned on in the definition)
	Access Policy – check for Request JWT in Auth Request
	FAPI Definition Configuration
	Access Policy – check for Request JWT in Auth Request

Procedure

1. **OIDC Definition**

   a) In the appliance dashboard, select **Federation** > **OpenID Connect and API**.

   b) In the **Definitions** tab, check the **OIDC Compliant** and **FAPI Compliant** check-box.

2. **WebSEAL OAuth Config**
   a) In the appliance dashboard, select **Web** > **Reverse Proxy**.
   b) Select a reverse proxy instance.
   c) Navigate to **Manage** > **AAC and Federation Configuration** > **OAuth and OpenID Connect Provider Configuration**.
   d) In the **Main** tab, check the FAPI compliant check-box.

Achieving OpenID Connect Provider conformance with IBM Security Verify Access

IBM Security Verify Access supports the OpenID Connect protocol. IBM Security Verify Access acts as both the OpenID Provider and the Relying Party.

This topic provides the information that is required to be performed on IBM Security Verify Access for it to be conformant as an OpenID Connect Provider.

Most of the scenarios are conformant Out-of-the-box. However there are some scenarios where access policies and mapping rule can be used.

These artifacts that are required to achieve conformance with IBM Security Verify Access are placed in a compressed file under **System** > **File Downloads** > **Federation** > **examples**. In the **examples** folder, download `oidc_op_conformance.zip` and extract its contents.

The `oidc_op_conformance.zip` contains the following files:

- `pre_token.js`
- `post_token.js`
- `authsvc_credential.js`
- `access_policy.js`
- `metadata.json`
- `httptransform.xsl`
- `stschains.json`

There are comments that are specified in the files listed above that explains in detail about the scenario that is achieved for OIDC Conformance.

The files also contain “OIDC Conformance-Example" which indicates a snippet of code to be added to achieve a certain scenario for conformance.

To achieve conformance on an existing IBM Security Verify Access setup, copy the snippets of AccessPolicy, Mapping Rule, and create the necessary STS chains.

OpenID Connect Provider Access Policies

You can use access policies to perform step-up and re-authentication during a single sign-on flow based on contextual information.

prompt and max_age

The access policy checks if one of the following attributes is requested from the authentication context and triggers an authentication policy based on the selected attribute:

- `max_age`
- `prompt=none`

For `max_age`, an `authenticationTime` attribute must be added to the `authsvc_credential` mapping rule. It is also available as an example.

The following prerequisites are required to use the access policy:

1. The access policy uses the Advanced Access Control, UsernamePassword policy. Click **AAC** > **Policy** > **Authentication** > **Mechanism**. Search for the **UsernamePassword** mechanism to configure it.
2. Change the ACL that is attached to `{junction}/sps/auth` from `anyauth` to `unauth` ACL.
3. The access policy that is created is to be used by the API Definition that is created.

Mapping Rules

authenticationTime

The `max_age` scenario requires an authentication time attribute which is added to the `authsvc_credential` mapping rule.

For more information, refer to the `oidc_op_conformance.zip` file for the following examples:

- **OIDC Conformance-Example 1.1.1** in the `authsvc_credential` mapping rule.
- **OIDC Conformance-Example 1.1.2** in the `pre_token` mapping rule.

produce_userinfo_jwt

One of the conformance scenarios is to be able to sign the `userinfo` response. This can be achieved in IBM Security Verify Access by sending `userinfo` response as a signed JWT.

To achieve conformance, a snippet of code is added to check if the `userinfo_signed_response_alg` is requested, based on that an STS chain is invoked to convert the `STSUniversalUser` to a signed JWT.

STS chain samples are available in the same zip file.

For more information, refer to the **OIDC Conformance-Example 1.2** in the `oidc_op_conformance.zip` file in the `post_token` mapping rule.

The mapping rule snippet retrieves the `userinfo_signed_response_alg` and uses `STSClientHelper` to call an STS chain.

redirect_uri

In IBM Security Verify Access we provide some flexibility for the `redirect_uri` check. Howerver, the OIDC conformance performs a strict check on the `redirect_uri` for it to be an identical match.

If the `redirect_uri` https://test.com/isam is registered in IBM Security Verify Access and request came in with a `redirect_uri` which is https://test.com/isam?example=one , IBM Security Verify Access flow succeeds.

However, OIDC conformance suggests that an error is thrown if it is not an exact match. To retain flexibility, IBM Security Verify Access made the choice to flag an error at the mapping rule. Hence the following snippet is added into the `pre_token` mapping rule to throw an error based on the `redirect_uri`.

For more information, refer to the **OIDC Conformance-Example 1.3** in the `oidc_op_conformance.zip` file in the `pre_token` mapping rule.

The mapping rule snippet retrieves the requested `redirect_uri` and compares it to the registered `redirect_uri`. It throws an error if an identical match is not found.

nonce

In OpenID connect specification, nonce is not a required parameter for `authorization_code` flow. In IBM Security Verify Access during an `authorization_code` flow, if a nonce is requested it is not returned as a claim in the id_token when the code is exchanged for token and id_token.

To enable this, see to the **OIDC Conformance-Example 1.4** in the `oidc_op_conformance.zip` file in the `pre_token` mapping rule. The mapping rule snippet retrieves the nonce during the code flow and during the token flow retrieves the nonce associated with the `state_id` and adds nonce as an `id_token` claim.

assert_no_code_reuse

The OAuth specification dictates that the Authorization server should revoke access tokens which are issued to a code, if that code is reused. In order to enable this enforcement a snippet in the `post_token` and the `pre_token` mapping rule must be enabled.

See **OIDC Conformance-Example 1.5.1** and **OIDC Conformance-Example 1.5.2**,the in pre_token and post_token mapping rule respectively.

STS Chains

Three STS chains are required to achieve conformance. The STS chain JSON is included in the compressed file.

Userinfo as JWT

This is used to generate a userinfo as a signed JWT. This chain is called from the post_token mapping rule.

The `appliesto` attribute must match `urn:appliesTo`.

The issuer must match `urn:issuer`.

The signing algorithm property for the JWT module can be set to RS256.

Request JWT (JWT to STSUU)

This STS chain is used to handle request and request_uri parameters. Parameters can be sent to the / authorize endpoint via a JWT or via a URL that contains the JWT.

This `appliesto` must match `https://localhost/sps/oauth/oauth20`.

The issuer must match REGEXP:`(urn:ibm:ITFIM:oauth20:client_request:.*)`.

See Passing parameters through JWT in a request to /authorize.

Client Authentication (JWT to STSUU)

This STS chain is used to handle Client Authentication using a JWT.

The `appliesto` must match `https://localhost/sps/oauth/oauth20`.

The issuer must match REGEXP:`(urn:ietf:params:oauth:client-assertion-type:jwt-bearer:.*)`.

See Client authentication to /token through an incoming JSON Web Token.

OpenID Connect Discovery

IBM Security Verify Access provides an endpoint for discovery which is the metadeta endpoint. However, the specification strictly states that the discovery endpoint is `/.well-known/openid-configuration` appended to the issuer endpoint.

We can use HTTP transformation rule to route requests to `/.well-known/openid-configuration` endpoint.

An example is attached with the compressed files.

For more information, refer to the **OIDC Conformance-Example 1.7** in the `oidc_op_conformance.zip` file.

Create an HTTP Transformation rule by using the `httptransform.xsl` and call it `httptransform`.

In the following example, API definition's issuer is configured to be `https://www.myidp.ibm.com/test`. The WebSEAL configuration must be updated.

```
[http-transformations]
updateuri = httptransform
[http-transformations:updateuri]
request-match = request:GET /test/.well-known/openid-configuration*
```

Also, there are some optional discovery parameters, which are required for conformance. To add these values, the `metadata.json` under **Federation** or **AAC**, **Template Files** can be modified.

The modification that must be made are as follows:

- Adding "claims_supported"with the following values:

  ```
  ["realmName","preferred_username","given_name","uid","upn","groupIds",
  "employee_id","name","tenantId","mobile_number","department","job_title",
  "family_name","email"]
  ```

- Adding "userinfo_signing_alg_values_supported" with this value: ["RS256"]
- Adding "request_parameter_supported" with the following value :

  ```
  <%var supported = true;templateContext.response.body.write(supported);%>
  ```

Note: For achieving key rotation in IBM Security Verify Access, we add a new key to the `rt_profile_keys` (which is used as a default keystore). Since the jwks lists keys to "use" during signing only, to list down the keys to "use" during encryption, we use the jwks_uri which is `https://<runtime_host>/sps/jwks`.

Achieving Financial-grade API (FAPI) conformance with IBM Security Verify Access

The Financial-grade API aims to provide specific implementation guidelines for online financial services.

The Financial-grade API security profile can be applied to online services in any market area that requires a higher level of security than provided by standard OAuth or OpenID Connect.

Verify Access supports the OpenID Connect protocol, ISAM can act both as an OpenID Provider and as a Relying Party. This topic provides additional information that is mentioned in the OpenID Connect Provider Conformance chapter.

Most of the scenarios are FAPI-conformant out-of-the-box. However there are some scenarios that require changes to be done to the mapping rules and transformation rules. To achieve FAPI conformance on an existing Verify Access setup, follow the procedures mentioned in this chapter.

Note: The steps mentioned in this document are additional procedures to perform in OpenID Connect Provider Conformance. Ensure that the procedures in the OpenID Connect Provider Conformance are completed before the procedures for FAPI conformance are performed.

OpenID Connect Discovery

There are some optional discovery parameters which are required to be present for FAPI conformance.

These information can be added to the `metadata.json` under **Federation or AAC** > **Template Files** > in order for the endpoint to be updated accordingly.

The following modifications must be made:

- Adding "`tls_client_certificate_bound_access_tokens`" to

  ```
  <%var supported = true;templateContext.response.body.write(supported);%>
  ```

- Updating "`claims_supported`" to

  ```
  ["realmName","preferred_username","given_name","uid","upn","groupIds","employee_id","name","tenantId",
  "mobile_number","department","job_title","family_name","email","acr"]
  ```

WebSEAL Configuration

As part of the FAPI conformance, the IBM Security Verify Access appliance supports Mutual TLS-based client authentication (MTLS) for confidential client authentication

About this task

To achieve FAPI MTLS on IBM Security Verify Access, perform the following tasks:

Disable TLS 1.0/1.1
As part of FAPI requirement, the appliance strictly disallows TLS 1.0/1.1 connections. See Step 1: Disable TLS 1.0/1.1

Only Allow Secure Cipher Suites
To ensure that IBM Security Verify Access uses only FAPI specification-compliant SSL version and ciphers for TLS Connection, see Step 2: Allow Secure Cipher Suites.

Procedure

1. To **disable TLS 1.0/1.1**, configure the WebSEAL configuration file by setting "`disable-tls-v1`" and "`disable-tls-v11`" to "`yes`".
   a) In the Appliance Dashboard, select **Web** > **Manage** > **Reverse Proxy**.
   b) Select the reverse proxy instance name and select **Manage** > **Configuration** > **Edit Configuration File**.
   c) In the configuration file, set `disable-tls-v1` and `disable-tls-v11` under "`yes`".
2. To **only allow secure cipher suites**:
   a) In the appliance dashboard, select **Web** > **Manage** > **Reverse Proxy**.
   b) Select the reverse proxy instance name.
   c) Select **Manage** > **Configuration** > **Edit Configuration File**.
   d) In the configuration file under **[ssl]**. disable tlsv11 and earlier:
      - disable-tls-v1 = yes
      - disable-tls-v11 = yes
   e) In the configuration file under **[ssl-qop-mgmt-default]**, set default ciphers to:
      - `default = TLS_DHE_RSA_WITH_AES_128_GCM_SHA256`
      - `default = TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256`
      - `default = TLS_DHE_RSA_WITH_AES_256_GCM_SHA384`
      - `default = TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384`
   f) In order for the appliance to use the DHE ciphers set in the previous step, a platform level flag must be set. This can be done with by setting `gsk-attr-name = enum:4009:1` under **[ssl]**.

HTTP Transformation Rules

Resource endpoint protection
As part of FAPI conformance, any resource endpoint that is protected by an OIDC API definition should support the following standards:

- Returns x-fapi-interaction-id
- ContentType: JsonUTF8

These requirements can be achieved by implementing the following HTTP transformation rule. The example assumes resource endpoint are files under a directory with the name 'resource'.

HTTP Transformation Rule

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsl:stylesheet xmlns:xsl="http://
www.w3.org/1999/XSL/Transform"
     version="1.0" xmlns:external="http://
xsltfunctions.isam.ibm.com">
     <xsl:strip-space elements="*" />
     <xsl:template match="/">
         <HTTPResponseChange>
             <xsl:apply-templates />
         </HTTPResponseChange>
     </xsl:template>
     <xsl:template match="//HTTPResponse/
Headers">
             <Header name="Content-
type" action="update">application/json;
 charset=utf-8</Header>
     </xsl:template>
     <xsl:template match="//HTTPResponse/
HTTPRequest/Headers/Header">
             <xsl:choose>
                 <xsl:when test="@name='x-
fapi-interaction-id' ">
                     <Header name="x-fapi-
interaction-id" action="add">
                         <xsl:value-of
 select="current()" />
                     </Header>
                 </xsl:when>
             </xsl:choose>
     </xsl:template>
 </xsl:stylesheet>
```

Reverse Proxy Configuration

```
[http-transformations]
 resourceReq=resourceReq
[http-transformations:resourceReq] request-
match = response:GET /resource*
 match-case-insensitive = yes
```

Mapping Rules

s_hash

IBM Security Verify Access does not support "s_hash" out-of-the-box. However, it can be calculated and added as claims by adding the following code snippet to pre_token mapping rules. See https://www.ibm.com/blogs/security-identity-access/openbanking-the-state-hash-claim/ for more information.

```
/*
FAPI - S_HASH
*/
importClass(Packages.java.util.Base64);

var request_type = null;
var grant_type = null;
var response_type = null;

// The request type - if none available assume 'resource'
var tmp = stsuu.getContextAttributes().getAttributeValuesByNameAndType("request_type",
 "urn:ibm:names:ITFIM:oauth:request");
if (tmp != null && tmp.length > 0) {
    request_type = tmp[0];
} else {
    request_type = "resource";
}

// The grant type
tmp = stsuu.getContextAttributes().getAttributeValuesByNameAndType("grant_type",
 "urn:ibm:names:ITFIM:oauth:body:param");
if (tmp != null && tmp.length > 0) {
    grant_type = tmp[0];
}

// The response type
tmp = stsuu.getContextAttributes().getAttributeValuesByName("response_type");
if (tmp != null && tmp.length > 0) {
    response_type = tmp[0];
}

var state = null;
```

```
if (request_type == "authorization" && response_type != null &&
 response_type.indexOf("id_token") > -1) {

    // When id_token to be produced at /authorize
    state = stsuu.getContextAttributes().getAttributeValueByName("state");

} else if (request_type == "access_token" && grant_type == "authorization_code") {

    // When id_token to be produced at /token
    var code = stsuu.getContextAttributes()
            .getAttributeValueByNameAndType("code", "urn:ibm:names:ITFIM:oauth:body:param");
    var token = OAuthMappingExtUtils.getToken(code);
    if (token != null) {
        state = OAuthMappingExtUtils.getAssociation(token.getStateId(), "state");
    }

}

if (state != null) {

    // Need to hash based on algorithm
    // The hash algorithm to use is dictated by the signing algorithm of JWT
    var alg = stsuu.getContextAttributes()
            .getAttributeValueByNameAndType("signing.alg", "urn:ibm:oidc10:jwt:create");

    // For now only SHA-256 and SHA-512 are supported natively by ISAM.
    // Consider using KJUR or similar if SHA384 is needed.
    var hash = null;

    if (alg != null) {
        if (alg.endsWith("256")) {
            hash = OAuthMappingExtUtils.SHA256Sum(state);
        } else if (alg.endsWith("384")) {
            hash = null; // Not supported!
        } else if (alg.endsWith("512")) {
            hash = OAuthMappingExtUtils.SHA512Sum(state);
        }
    }

    if (hash != null && hash.length > 0) {
        var state_hash =
 Base64.getUrlEncoder().withoutPadding().encodeToString(hash.splice(0, hash.length/2));
        var attr = new com.tivoli.am.fim.trustserver.sts.uuser.Attribute("s_hash",
 "urn:ibm:jwt:claim", state_hash);
        stsuu.addAttribute(attr);
    }

}

/*
FAPI - S_HASH
*/
```

Response type code

As part of FAPI conformance, response type code is not permitted. Therefore incoming request with response type code throws an unsupported error page. This can be achieved by including the following snippet in pre_token mapping rules.

```
/*Disallow Code Flow*/
var response_type= null;
if (request_type="authorization"){
    temp_attr =
 stsuu.getContextAttributes().getAttributeValuesByNameAndType("response_type",
 "urn:ibm:names:ITFIM:oauth:query:param");
    if (temp_attr != null && temp_attr.length > 0) {
            response_type = temp_attr[0];
    } else {
        temp_attr =
 stsuu.getContextAttributes().getAttributeValuesByNameAndType("response_type",
 "urn:ibm:names:ITFIM:oauth:body:param");
        if (temp_attr != null && temp_attr.length > 0) {
            response_type = temp_attr[0];
            IDMappingExtUtils.traceString("response_type :::"+response_type );
        }
    }
    if ("code" == response_type) {
        //IDMappingExtUtils.throwSTSException("Code Flow Disallowed");
        OAuthMappingExtUtils.throwSTSCustomUserPageException("Unsupported response type
 code.",400,"invalid_request");
```

```
    }
}
/*Disallow Code Flow*/
```

Disallowing state in request parameter

As part of FAPI conformance, only claims passed in request object is used and returned. This rule ensures that when state is passed in request parameter, the appliance does not take it into consideration during the flow by removing it from the stsuu in the pre_token mapping rule.

```
/*Disallow State in Request Param*/
var reqParam_state =
 stsuu.getContextAttributes().getAttributeValueByNameAndType("state","urn:ibm:names:ITFIM:oauth:query:param");
if (reqParam_state != null){
var x = stsuu.getContextAttributes().removeAttributeByNameAndType("state","urn:ibm:names:ITFIM:oauth:query:param");
}
/*Disallow State in Request Param*/
```

STS Chains

As a part of OIDC OP Conformance, an STS chain is created to handle parameters that are sent in a JWT to the authorize endpoint. This STS chain template must be updated to include a map module.

UpdateRequestJWT (JWTtoMaptoSTSUU)

This STS chain is used to handle request and request_uri parameters. Parameters can be sent to the `/authorize` endpoint by using a JWT or by using a URL that contains the JWT.

The applies to must match `https://localhost/sps/oauth/oauth20`.

The issuer must match REGEXP:`(urn:ibm:ITFIM:oauth20:client_request:.*)`. See Passing parameters through JWT in a request to /authorize.

The map module performs basic request object checks which are required for FAPI conformance. This map module links to the following mapping rule that can be uploaded into mapping rules in the appliance.

```
importPackage(Packages.com.tivoli.am.fim.trustserver.sts);
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.oauth20);
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.uuser);
importPackage(Packages.com.ibm.security.access.user);
importClass(Packages.com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils);
importClass(Packages.com.tivoli.am.fim.trustserver.sts.utilities.OAuthMappingExtUtils);
importClass(Packages.com.ibm.security.access.httpclient.HttpClient);
importClass(Packages.com.ibm.security.access.httpclient.HttpResponse);
importClass(Packages.com.ibm.security.access.httpclient.Headers);
importClass(Packages.com.ibm.security.access.httpclient.Parameters);
importClass(Packages.java.util.ArrayList);
importClass(Packages.java.util.HashMap);
var claims_str = stsuu.getContextAttributes().getAttributeValueByNameAndType("claim_json",
 "urn:com:ibm:JWT");
var claims = JSON.parse(claims_str);
var header_str = stsuu.getContextAttributes().getAttributeValueByNameAndType("header",
 "urn:com:ibm:JWT");
var headers = JSON.parse(header_str);
/*
 * Checks that request object contains exp, scope, nonce, redirect_uri.
 */
requestObjPass = true
if ( claims.exp == undefined){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("exp is missing in request
 object.",400,"invalid_request");
}
if ( claims.scope == undefined ){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("scope is missing in request
 object.",400,"invalid_request");
}
if ( claims.nonce == undefined ){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("nonce is missing in request object.
 ",400,"invalid_request");
}
if (claims.redirect_uri == undefined){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("redirect_uri in request object is
 missing. ",400,"invalid_request");
}
if (headers.alg == "none"){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("alg in request object value cannot
 be none. ",400,"invalid_request");
}
/*
 * Check that the JWT has not expired
 */
if ( claims.exp != undefined ){
    var expDate = new Date(claims.exp * 1000);
    var currDate = new Date();
    if (expDate < currDate){
        OAuthMappingExtUtils.throwSTSCustomUserPageException("Request object has
 expired.",400,"invalid_request");
   }
}
/*
 * Validates aud and issuer value in request object against information in definition.
 */
if ( claims.iss != undefined ){
    var defID = OAuthMappingExtUtils.getClient(claims.iss).getDefinitionID();
    var iss = OAuthMappingExtUtils.getDefinitionByID(defID).getOidc().getIss();
    if (Array.isArray(claims.aud)){
       var found = false;
       for (var x = 0; x < claims.aud.length; x++ ){
           if( claims.aud[x]!= iss ){
                found = true;
           }
       }
       if (!found){
          OAuthMappingExtUtils.throwSTSCustomUserPageException("aud in request object does
 not match issuer of client definition.",400,"invalid_request");
       }
    }
    else if( claims.aud != iss ){
        OAuthMappingExtUtils.throwSTSCustomUserPageException("aud in request object does not
 match issuer of client definition.",400,"invalid_request");
    }
}
/*
 * Ensures Nonce/State length are within supported range, 255.
 */
if ( claims.state != undefined && claims.state.length > 255){
    OAuthMappingExtUtils.throwSTSCustomUserPageException("State in request object exceeds
 supported limit.",400,"invalid_request");
}
if ( claims.nonce != undefined && claims.nonce.length > 255){
    OAuthMappingExtUtils.throwSTSCustomUserPageException("Nonce in request object exceeds
 supported limit.",400,"invalid_request");
}
```

FAPI Definition Configurations

This topic describes the FAPI definitions configurations.

Definition Configuration- Minimum Entropy 128bit
: As part of FAPI requirement, access_tokens should be a minimum of 128 bit. To configure this set `access_token length` to '32' in OpenID Definition.

Definition & Advanced Configuration - Update HTML Encoded Macro
: As part of FAPI requirement, users are allowed to reject login attempt upon authentication. This can be achieved by setting `Prompt` to `Always allow` in the Definition configuration. This would mean that the prompt page is triggered during the SSO flow after the user successfully logs in. In order for claims passed to be successfully added without being html encoded you can add the macro `@OAUTH_OTHER_PARAM_VALUE_REPEAT@` to `sps.page.htmlEscapedMacros`.

Definition Configuration- Use EC256 or PS256 Signing keys
: As part of FAPI requirement, ES256 signing keys are required to be used for `id_token` signing.

Advance Configuration- OAuth20.State.Required
: FAPI conformance requires authorization requests without state to be allowed. To achieve this, set the advance configuration parameter `OAuth20.State.Required` to `false`. The configuration overwrites IBM Security Verify Access default behavior making state a non-mandatory parameter.

Note: This can only be achieved on IBM Security Verify Access version 10.0.0

FAPI - MTLS and Certificate Bound Tokens

FAPI specs require that Verify Access supports [OAUTB] or [MTLS] as a hold of key mechanism.

This specification requires clients to authenticate to token endpoint or resource endpoint with a client certificate. The authenticated client certificate is then bound to tokens (access_token, refresh_token, and code) that are generated for the client.

FAPI- Certificate Authentication and MTLS

To handle Client Cert Authentication according to FAPI Specs, Cert-EAI can be used. The following are steps to configure this.

Creating Authentication Mechanism (available by default from IBM Security Verify Access version 10.0.0)

1. Create an InfoMap Authentication Mechanism with `FAPI_CertEAI` as Mapping rule and `cert_mismatch.json` as template page.
2. Create an Authentication Policy with the authentication mechanism that is created from the previous step.

`/authsvc/authenticator/Infomap/cert_mismatch.json`

```
{
"Error": "Certificate Mismatch"
<% templateContext.response.setHeader("am-eai-flags", "stream");
templateContext.response.setHeader("Content-Type", "application/json");
templateContext.response.setStatus(401);
%>
}
```

Configuring CertEAI in WebSEAL

Configure CertEAI to call authentication policy that is created in "Creating Authentication Mechanism (available by default from IBM Security Verify Access version 10.0.0)" on page 115 and pass the certificate data. See the following example:

```
[certificate]
accept-client-certs = optional
eai-uri = junction+"/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:fapi_CertAuth"
eai-data = Base64Certificate:cert
eai-data = SubjectCN:SubjectCN
eai-data = Fingerprint:Fingerprint
```

Note: Client Certificate must be added to WebSEAL's SSL DB (`pdsrv`) and ensure that *junction* has an appropriate value

FAPI Certificate Authentication EAI

FAPI_CertEAI, the infomap that is used in FAPI Authentication Mechanism, checks if certificate is bound to token matches the certificate in incoming request.

FAPI_CertEAI

```
importClass(Packages.com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils);
importClass(Packages.com.ibm.security.access.user.UserLookupHelper);
importPackage(Packages.com.ibm.security.access.httpclient);
importClass(Packages.com.tivoli.am.fim.trustserver.sts.utilities.OAuthMappingExtUtils);

IDMappingExtUtils.traceString("Entering FAPI Infomap");

var cert = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header", "cert");
var subjectCN = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header",
 "SubjectCN");
var fingerprint = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header",
 "Fingerprint");

if (cert != null && subjectCN != null && fingerprint!= null){
 // useful trace
 IDMappingExtUtils.traceString(" cert: " + cert);
 IDMappingExtUtils.traceString(" SubjectCN: " + subjectCN);
 IDMappingExtUtils.traceString(" fingerprint: " + fingerprint);
```

```
  var auth_header = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header",
  "Authorization");
  //Check if there is a authorization header
  if(auth_header != null){
   var array = auth_header.split(" ");
   if (array[0].equals("Bearer")){
    var token = array[1];

    //Introspect the access_token
    var tkn = OAuthMappingExtUtils.getToken(token)
    var cnf = OAuthMappingExtUtils.getAssociation(tkn.getStateId(), "cnf")
    IDMappingExtUtils.traceString("cnf:: [" + cnf + "]");
    if (cnf != fingerprint){
     IDMappingExtUtils.traceString("MISMATCHED CERTS");
     success.setValue(false);
    }
    else{
     var iv_user_l = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header",
  "iv-user-l");
     context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes",
  "username", subjectCN);
     context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes", "cert",
  cert);
     context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes",
  "fingerprint", fingerprint);
     success.setValue(true);
    }
   }
  }
  else{
   var iv_user_l = context.get(Scope.REQUEST, "urn:ibm:security:asf:request:header", "iv-
 user-l");
   context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes",
  "username", subjectCN);
   context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes", "cert",
  cert);
   context.set(Scope.SESSION, "urn:ibm:security:asf:response:token:attributes",
  "fingerprint", fingerprint);
   success.setValue(true);
  }
 }
 else{
  IDMappingExtUtils.traceString("Certificate information unavailable");
  success.setValue(false);
 }
```

Certificate bound token can be achieved by associating token’s state ID to the certificate thumbprint upon token creation in `oauth post_token` mapping rule. When these tokens are used again at resource endpoint, a check is done in the pre_token mapping rule to retrieve the associated thumbprint and check if it matches the incoming certificate thumbprint of the client. If it matches, access is allowed to the resource. If it does not match, an error is thrown. See the following example:

When there is an issue with certificate bound token and an error is thrown. WebSEAL returns a `stepuplogin` error. In FAPI, this error is a 401 Unauthorized or `Invalid_request`. To achieve this error on IBM Security Verify Access, create an error file (Select **Reverse Proxy** > **Manage** > **Management Root**) . Under **management/C/**, create `stepuplogin.401.json`. This returns a 401 error. The following is a sample content of the file:

```
{
  "error_code"  : " Unauthorized "
  "error_message" : " Client Certificate Mismatch. This resource can only be access by an
 authorized user. "
  }
```

FAPI specs requires resource endpoint to be protected by mtls. This can be done by enforcing a ‘`ext-auth-interface`’ authentication level by using a Protected Object Policy (POP).

```
[session]
require-mpa = yes
```

Setting `require-mpa` to 'yes' means that HTTP headers are not valid session keys or authentication tokens unless received through an MPA. In FAPI, this functionality is used to ensure each token and the cert information are build as one unique session without any form of session caching.

FAPI- Certificate Bound Token

At the resource endpoint, a check is done in the `pre_token` mapping rule to retrieve the associated thumbprint and checks if the thumbprint matches the incoming certificate thumbprint of the client. If the thumbprint matches, access is allowed to the resource. If the thumbprint does not match, an error is thrown. See the following example:

oauth20 pre_token mapping rule

```
/*Cert Bound Tokens*/
 if (request_type == "resource"){
 var incoming_thumbprint =
 stsuu.getContextAttributes().getAttributeValueByName("tagvalue_x509fingerprint");
 var incoming_access_token =
 stsuu.getContextAttributes().getAttributeValueByName("access_token");
 var incoming_refresh_token =
 stsuu.getContextAttributes().getAttributeValueByName("refresh_token");
 var incoming_code = stsuu.getContextAttributes().getAttributeValueByName("code");


 if (incoming_access_token != null && incoming_thumbprint != null){
 var state_id = OAuthMappingExtUtils.getToken(incoming_access_token).getStateId();
 var original_thumbprint = OAuthMappingExtUtils.getAssociation(state_id, "cnf");

 if (original_thumbprint != incoming_thumbprint){
     OAuthMappingExtUtils.throwSTSCustomUserPageException("INCOMING MTLS client cert does
 not match access_token client's cert",400,"invalid_request");
 }

 }else if (incoming_refresh_token != null && incoming_thumbprint != null){
 var state_id = OAuthMappingExtUtils.getToken(incoming_refresh_token).getStateId();
 var original_thumbprint = OAuthMappingExtUtils.getAssociation(state_id, "cnf");

 if (original_thumbprint != incoming_thumbprint){
     OAuthMappingExtUtils.throwSTSCustomUserPageException("INCOMING MTLS client cert does
 not match access_token client's cert",400,"invalid_request");
 }

 }else if (incoming_code != null && incoming_thumbprint != null){
 var state_id = OAuthMappingExtUtils.getToken(incoming_code).getStateId();
 var original_thumbprint = OAuthMappingExtUtils.getAssociation(state_id, "cnf");

 if (original_thumbprint != incoming_thumbprint){
     OAuthMappingExtUtils.throwSTSCustomUserPageException("INCOMING MTLS client cert does
 not match access_token client's cert",400,"invalid_request");
 }

 }
 }
 /*Cert Bound Tokens*/
```

oauth20 post_token mapping rule

```
/*
FAPI - Cert Bound Tokens
*/
var client_thumbprintAttribute = stsuu.getAttributeValueByName("tagvalue_x509fingerprint");

if (state_id != null && client_thumbprintAttribute != null){
   var result = OAuthMappingExtUtils.associate(state_id, "cnf",
client_thumbprintAttribute);
}
/*
FAPI - Cert Bound Tokens
*/
```

FAPI- Private Key JWT

When `FAPI_CertEAI` authenticates a client with MTLS, `client_assertion` STS chain is not triggered as the client is already authenticated.

The following code snippet triggers `client_assertion` STS Chain. This ensures client present in `client_assertion` and MTLS authenticated client matches.

To achieve FAPI certification with Private Key JWT, add the following snippet to the API Protection Definition `pre token` mapping rule:

```
/*
 * FAPI - Private_key_jwt
 *
 * Ensure this snippet is added within the isFapiCompliantByDefinitionID check
 * Client Assertion is a form of client authentication. In the case of FAPI, FAPI_CertEAI
 authenticates a client if mtls client certificate is present.
 * Therefore, client_assertion is not handled. This code snippet triggeres client_assertion
 manually and ensures client id of client_assertion jwt and client object that was created
 * based of mtls authentication matches.
 *
 * client_assertion_required flag enforces client_assertion.
 *
 */
var client_assertion_required = true;
var client_assertion = stsuu.getContextAttributes().getAttributeValueByName("client_assertion");
if (client_assertion_required && request_type == "access_token" && client_assertion != null){
 var base_token = IDMappingExtUtils.stringToXMLElement(
   "<wss:BinarySecurityToken "
 + "xmlns:wss=\"http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd\" "
 + "wss:EncodingType=\"http://ibm.com/2004/01/itfim/base64encode\" "
 + "wss:ValueType=\"urn:com:ibm:JWT\">"+client_assertion+"<\/wss:BinarySecurityToken>");
 var res = LocalSTSClient.doRequest("http://schemas.xmlsoap.org/ws/2005/02/trust/
Validate","https://localhost/sps/oauth/oauth20", "urn:ietf:params:oauth:client-assertion-
type:jwt-bearer:", base_token, null)
    if (res.errorMessage == null){
        var client_assertion_stsuu = new STSUniversalUser();
     client_assertion_stsuu.fromXML(res.token);
  IDMappingExtUtils.traceString("FAPI Client Assertion Result: " + client_assertion_stsuu);

     var claims_str =
 client_assertion_stsuu.getContextAttributes().getAttributeValueByNameAndType("claim_json",
 "urn:com:ibm:JWT");
     var claims = JSON.parse(claims_str);

     /*
      * Check that the JWT has not expired
      */
     if ( claims.exp != undefined ){
    var expDate = new Date(claims.exp * 1000);
    var currDate = new Date();
    if (expDate < currDate){
      OAuthMappingExtUtils.throwSTSCustomUserPageException("JWT has
 expired.",400,"invalid_request");
     }
     }

  /*
   * Validates aud and issuer value in client_assertion jwt against information in definition.
   */
  if ( claims.iss != undefined && claims.aud != undefined){
   var def_id = OAuthMappingExtUtils.getClient(claims.iss).getDefinitionID();
   var iss = OAuthMappingExtUtils.getDefinitionByID(def_id).getOidc().getIss();
   var poc = OAuthMappingExtUtils.getDefinitionByID(def_id).getOidc().getPoc();
   if (Array.isArray(claims.aud)){
      var found = false;
      for (var x = 0; x < claims.aud.length; x++ ){
       if((claims.aud[x]).includes(iss) || (claims.aud[x]).includes(poc)){
      found = true;
      break;
       }
      }
      if (!found){
      OAuthMappingExtUtils.throwSTSCustomUserPageException("aud in request object does not match
 issuer of client definition.",400,"invalid_request");
      }
   }
   else if( !((claims.aud).includes(iss) || (claims.aud).includes(poc))){
    OAuthMappingExtUtils.throwSTSCustomUserPageException("aud in client_assertion jwt does not
 match issuer of client definition.",400,"invalid_request");
   }
     }

  /*
   * Ensure MTLS authentication credentials present
   */
  var fingerprint = stsuu.getAttributeValueByName("fingerprint");
  if (fingerprint == null){
   OAuthMappingExtUtils.throwSTSCustomUserPageException("mtls credentials of client is
 missing.",400,"invalid_request");
  }

 }else{
  OAuthMappingExtUtils.throwSTSCustomUserPageException("client_assertion
 failed.",400,"invalid_request");
 }
}else if(client_assertion_required && request_type == "access_token" ){
  OAuthMappingExtUtils.throwSTSCustomUserPageException("client_assertion is not found in token
 endpoint.",400,"invalid_request");
}
```

Configuring FAPI Client

FAPI conformance requires MTLS and Certificate bound token to use a Client Certificate.

You can bind a certificate that is added to the trust store, to a client. To bind a certificate, add the client certificate details (for example, alias and keystore) to the extended properties when you are creating a client. This can be achieved by navigating to **Federation** > **OpenID Connect and API Protection** > **Clients**. This can also be done for dynamic clients.

```
{
  "tls_client_auth_subject_dn": "clientID",
  "tls_client_auth_keystore": "rt_profile_keys "
}
```

The information that is added to client configuration can then be used to verify if the incoming mtls certificate matches client certificate. Use the following code snippet at `FAPI_ValidateJWT_RequestJWT` mapping rule or `oauth20_pre_token` mapping rule to verify:

```
/*
 * Certificate and Jwt signing key check
 * claims.iss can be substituted with client id
 * headers.kid can be substituted with fingerprint
 (stsuu.getAttributeValueByName("fingerprint");)
 * Please note that (stsuu.getAttributeValueByName("fingerprint");) returns thumbprint in
 OAuthMappingExtUtils.getCertificateThumbprint format.
 */
var client_ExtendedData = OAuthMappingExtUtils.getClient(claims.iss).getExtendedData();
if ( client_ExtendedData != null){
 var client_keystore = JSON.parse(client_ExtendedData).dynamic_client.tls_client_auth_keystore;
 var client_alias = JSON.parse(client_ExtendedData).dynamic_client.tls_client_auth_subject_dn;
 if (client_alias != null && client_keystore != null){
    var cert_thumbprint =
 OAuthMappingExtUtils.getCertificateThumbprint_S256(client_keystore,client_alias);
    if (cert_thumbprint != null && cert_thumbprint != headers.kid){
     OAuthMappingExtUtils.throwSTSCustomUserPageException("Client certificate mis-
match!!!",400,"invalid_request");
    }
 }
}
```

Chapter 6. Configuring STS modules

Configure Security Token Service (STS) modules to validate and exchange security token types.

About this task

The STS is a component of the federation runtime that accepts WS-Trust requests for the validation and exchange of one security token type for another. You can configure the STS artifacts, which consist of modules, templates, and chains. These configuration elements allow an incoming WS-Trust message to be mapped to a particular template and its configuration.

These steps apply to the configuration for all of the "Supported module types" on page 121.

Procedure

1. Configure the token module prerequisites.
   - The Attribute Mapping module requires that you set up attribute sources. See Managing attribute sources.
   - The Username Token module requires that you set up server connections. See Managing server connections
   - The LTPA module requires that you import the LTPA key file. See Managing LTPA keys.
   - The Default Mapping module requires that you import the JavaScript rule file. See Managing JavaScript mapping rules.
2. View the module instances that are available. See Managing modules.
3. Create a new template or use an existing one. See Managing templates.
4. Create a new module chain. See Managing module chains.
5. Configure the module properties within the chain. Use the **Properties** tab inside of the module chain for the module.

Supported module types

STS modules are assembled as part of an STS chain that issues and validates specific types of tokens. IBM Security Verify Access supports several STS module types.

Attribute Mapping module

The Attribute Mapping STS module injects attribute values from different sources into an STSUU. This method to add attributes into the STSUU is convenient if you do not know how to write a mapping rule.

The Attribute Mapping module is called `AttributeMappingModule`.

Before using the attribute mapping module, you must configure the attribute sources so that they are available for selection. See Managing attribute sources.

Scenarios

- Single sign-on federations
- Custom trust chains

Supported modes

- Map

Configuration properties

Attribute Name
The attribute name that is populated into the STSUniversalUser.

Attribute Source
The name of a configured attribute source object. See Managing attribute sources for information about configuring attribute sources.

Default Mapping module

The Default Mapping module facilitates mapping by using an identity mapping rule.

The Default Mapping module is called `XSLTransformationModule`. The default mapping configuration consists of a JavaScript file that specifies an identity mapping rule. See Managing JavaScript mapping rules.

The module calls a JavaScript engine to read and run the identity mapping rules to generate a Secure Token Service Universal User (STSUU) XML document. The generated STSUU XML document contains the user identity information.

Scenarios

- Single sign-on federations
- Custom trust chains

Supported modes

- Map

Configuration properties

JavaScript file containing the identity mapping rule
The ID of the JavaScript file that contains the identity mapping rule.

For example, enter 8.

You must complete the mapping rule file and upload it before you can configure it into the chain.

HTTP Callout module

The HTTP callout module invokes a web service and enriches the STSUU with the returned contents.

Scenarios

- Single sign-on federations
- Custom trust chains

Supported modes

- Map

Configuration properties

Identify the URI format
The URI scheme.

HTTP
Use `http` for resources that are not protected by SSL.

HTTPS
Use `https` for resources that are protected by SSL.

Provide the web service URI
The endpoint address of the web service.

Server Certificate Database
The trust store containing the certificate of the HTTPS URL to call out to. This option is only required if the URI format is HTTPS.

Client authentication type
Specify the type of authentication to use:

No authentication
No credentials are required.

Basic authentication
Supply the basic authentication credentials:

Username
Specify the user name.

Password
Specify the password.

Client certificate authentication
Authenticate using a client certificate

Select the message format to use

XML
Use XML format for the message.

WS-Trust
Use WS-Trust format for the message.

IVCred module

The Verify Access credential module creates and consumes Verify Access-specific credentials. These credentials are called *IVCreds*.

The IVCred token module is called `IVCredModule`.The trust service can create and use local tokens in an environment that is protected by Verify Access. The support for Access Manager credentials means that the trust service can also use the credentials for authorization decisions.

Supported modes

- Validate
- Issue

Configuration properties

Validate mode

Enable signature validation
Enables or disables validation of signatures in the token module. Select the check box to enable signature validation.

Select validation key
Specifies the validation key that the partner must use.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Issue mode

List the attribute types to include

Specifies the attribute type of the attributes to be inserted during token creation. The attributes consist of information about the identity (user).

By default, all types are supported, as indicated by the asterisk (*) wildcard character.

Enable signatures
Specifies that signatures must be added to tokens.

Select the signing key
Specifies the key to use to sign tokens.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Select the KeyInfo elements to include
Specifies the elements of the signing certificate in the extended attributes of the credential. These attributes are only included if signatures are enabled. The default is for them to be disabled.

Public Key
Select to include the public key. If selected, the public key of the signing certificate is included in the Base64 encoded form. The extended attribute is labeled `ITFIM_IVCRED_SIGNER_CERTIFICATE_PUBKEY`.

Clear the check box to exclude the public key.

X509 Subject Name
Select to include this attribute. If selected, the distinguished name of the subject for the signing certificate is included. The extended attribute is labeled `ITFIM_IVCRED_SIGNER_CERTIFICATE_SUBJECT`.

Clear the check box to exclude the X509 Subject Name.

X509 Subject Issuer Details
Select to include this attribute. If selected, the issuer details of the signing certificate are included. The extended attribute is labeled `ITFIM_IVCRED_SIGNER_CERTIFICATE_ISSUER`.

Clear the check box to exclude the X509 Subject Issuer Details.

X509 Subject Key Identifier
Select to include this attribute. If selected, the subject key identifier of the signing certificate is included. The extended attribute is labeled `ITFIM_IVCRED_SIGNER_CERTIFICATE_SKI`.

Clear the check box to exclude the X509 Subject Key Identifier.

X509 Certificate Data
Select to include this attribute. If selected, the certificate data of the signing certificate is included in the Base64 encoded form. The extended attribute is labeled `ITFIM_IVCRED_SIGNER_CERTIFICATE`.

Clear the check box to exclude the X509 Certificate Data.

Note: If none of the `KeyInfo` elements are selected, `X509Certificate` data is still included in the signature by default.

LTPA module

The LTPA module facilitates the validating and issuing of LTPA version 1 and version 2 tokens.

The LTPA module is called `STSLTPATokenModule`.

An LTPA token is an encrypted string that contains user information and other metadata. Version 1 tokens contain fairly limited information, such as username and token expiration time. Version 2 tokens are extensible in that they can contain user-defined attributes, where each attribute can contain a list of values.

These tokens are represented as `BinarySecurityToken` elements.

This module does not support the initial generation of LTPA keys. You must provide a set of LTPA keys that were generated by another source such as a WebSphere® application server.

Supported modes

- Validate
- Issue

Configuration properties
Validate mode

LTPA file
Select the LTPA file to use.

You must upload the LTPA file into `/wga/ltpa_key` first for it to display in the list.

Password for key protection
(Required) The password that was used to protect the keys that are created by the partner.

Use the FIPS standard
Select to enable the Federal Information Processing Standards (FIPS). If FIPS was enabled when you created your partner, select this check box. The default is unchecked.

Issue mode

LTPA file
Select the LTPA file to use.

You must upload the LTPA file into `/wga/ltpa_key` first for it to display in the list.

Password for key protection
(Required) The password that was used to protect the keys that are created by the partner. It must be the same password that was used when the keys were created by the partner.

Use the FIPS standard
Select to enable the Federal Information Processing Standards (FIPS). If FIPS was enabled when you created your partner, select this check box. The default is unchecked.

Number of minutes before the created token expires
(Required) Indicates how long, from the time of token creation, the token remains valid. Specify the value in minutes. You can override this value by using the expiration Principle value in the Universal User. The default value is `120` minutes.

Realm used to create the user ID
The realm name to append to the user ID during token creation. You can override this value by using the realm Principle value in the Universal User. If you do not specify a name here, then the realm from the imported LTPA file is assumed.

Version of LTPA token to issue
The version number of the LTPA token you are issuing. Select 1 or 2 from the list, denoting LTPA Version 1 or Version 2.

Attributes to add to a version 2 token

Specify the type of attributes to include in the assertion. Use this field only for LTPA Version 2 tokens. An asterisk (*) indicates that all of the attribute types that are specified in the identity mapping file are included in the assertion.

To specify one specific type individually, type the attribute type in the text box. For example, if you want to include only attributes of type `urn:oasis:names:tc:SAML:2.0:assertion` in the assertion, type that string in the text box.

SAML 2.0 module

The SAML 2.0 module validates and issues SAML 2.0 tokens. This module is used for single sign-on in SAML 2.0 federations.

The SAML 2.0 module is called `Saml20STSTokenModule`.

Security Assertion Markup Language 2.0 (SAML 2.0) is a version of the SAML standard for exchanging authentication and authorization data between security domains. SAML 2.0 enables web-based authentication and authorization scenarios including cross-domain single sign-on (SSO), which helps reduce the administrative overhead of distributing multiple authentication tokens to the user.

Scenarios

- Single sign-on federations
- Custom trust chains

Supported modes

- Validate
- Issue
- Exchange

Configuration properties
Validate mode

Enable one-time assertion use enforcement
Specifies whether to use the assertion or token only once.

Enable signature validation
Enables or disables validation of signatures in the token module. Even if you do not select the check box, you must provide the key for decryption.

Select a validation key
Specifies the validation key that the partner must use.

Use the KeyInfo of the XML signature to find the X509 certificate for signature validation
Determines the appropriate certificate for signature validation. When you select this option, you must provide the subject distinguished name that matches the certificate.

RegExp
Specifies a regular expression to validate the subject distinguished name returned in the`KeyInfo`.

Use the keystore alias to find the public key for signature validation
Specifies a public key for signature validation, which is the default. Select the certificate database and label.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Select a decryption key
Select the key to use to decrypt encrypted messages.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Create multiple attribute statements in the Universal User
Specifies whether to keep multiple attribute statements in the groups in which they were received. This option might be necessary if your custom identity mapping rules are written to operate on one or more specific groups of attribute statements.

If you do not select this check box, multiple attribute statements are arranged into a single group (AttributeList) in the `STSUniversalUser` document. The default setting of the check box is not selected. This setting is appropriate for most configurations.

Map unknown name identifiers to the anonymous username
Specifies that the service provider can map an unknown persistent name identifier alias to the anonymous user account. By default, this option is disabled.

Default NameID format for assertion validation
Specifies a parameter for use during validation of a SAML assertion. The parameter is used to determine processing rules for the NameID element when one of the following conditions exists:

- If there is not an explicit Format attribute that is included in the assertion
- If the Format attribute is `urn:oasis:names:tc:SAML:2.0:nameid-format:unspecified`

Typically this parameter is needed only for STS chains that process SAML assertions that do not set the Format attribute. A normal example value is `urn:oasis:names:tc:SAML:2.0:nameid-format:emailAddress`.

Issue and Exchange mode

Name of the organization issuing the assertions
Shows a string that specifies the name of the organization (for example, a company) that issues the SAML assertions.

Amount of time before the issue date that an assertion is considered valid (seconds)
Default: `60` seconds

There is no minimum or maximum value enforced.

This field must contain a value.

Amount of time that the assertion is valid after being issued (seconds)
Default: `60` seconds

There is no minimum or maximum value enforced.

This field must contain a value.

List the attribute types to include
Specifies the types of attributes to be inserted during token creation. The attributes consist of information about the identity (user). Use && to separate attribute types. By default, all types are supported, as indicated by the asterisk (*) wildcard character.

For example, to add user-defined attribute types `type1` and `type2`, enter:

```
type1&&type2
```

Sign SAML assertions
Select if SAML assertions must be signed. Even if you do not select the check box, you must provide the key for encryption assertions.

Select the key for signing assertions
Specifies the key to use when signing SAML assertions.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Select the KeyInfo elements to include
Determines what `KeyInfo` elements to include in the digital signature when signing a SAML message or assertion. Select one or more of the following elements.

X509 Subject Key Identifier
Select to include the X.509 subject key identifier with your signature. If not selected, the subject key identifier is excluded. To change the default for this element, change it in the custom properties.

Public Key
Select to include the public key with your signature. If not selected, the public key is excluded. To change the default for this element, change it in the custom properties.

X509 Subject Issuer Details
Select to include the issuer name and the certificate serial number with your signature. If not selected, the subject issuer details are excluded. To change the default for this element, change it in the custom properties.

X509 Subject Name
Select to include the X.509 subject name with your signature. If not selected, the X.509 data is excluded. To change the default for this element, change it in the custom properties.

X509 Certificate Data
Select to include the BASE64 encoded certificate data with your signature. If not selected, the X.509 data is excluded. To change the default for this element, change it in the custom properties.

Note: If you do not select any of the `KeyInfo` elements, X.509 certificate data is still included in the signature by default.

Signature algorithm for signing SAML assertions
Specifies the signature algorithm to use to sign the SAML assertion.

RSA-SHA1
`http://www.w3.org/2000/09/xmldsig#rsa-sha1`

DSA-SHA1
`http://www.w3.org/2000/09/xmldsig#dsa-sha1`

RSA-SHA256
`http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`

Note: The chosen signature algorithm must match the signing key type that was set in the federation level to prevent a signature failure. For example, select DSA-SHA1 for DSA keys.

Select the key for encrypting assertion elements for this partner
Specifies the key to use to encrypt assertions.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Encrypt assertions
Specifies whether assertions are to be encrypted. If selected, specify an encryption key.

Encrypt assertion attribute elements
Specifies whether Attribute elements within the assertions are to be encrypted. If selected, specify an encryption key.

Encrypt NameID elements in assertions
Specifies whether `NameID` elements in the assertions are to be encrypted. If selected, specify an encryption key.

Block encryption algorithm
Specifies the encryption algorithm to use to encrypt data for this partner.

Triple DES
Triple Digital Encryption Standard

AES-128
Advanced Encryption Standard 128-bit

AES-192
Advanced Encryption Standard 192-bit

AES-256
Advanced Encryption Standard 256-bit

Subject confirmation method
Specifies the subject confirmation method for the assertion. You can select one or more subject confirmation methods at the same time, or choose not to select any confirmation methods. If you select the holder-of-key type, the default includes the X.509 Certificate Data in the `KeyInfo` for the `SubjectConfirmationMethod`. `STSUniversalUser` can provide the data for the subject confirmation method `KeyInfo`. The data can also be extracted from the signed request data.

Valid values can be:

- `urn:oasis:names:tc:SAML:2.0:bearer`
- `urn:oasis:names:tc:SAML:2.0:holder-of-key`

- `urn:oasis:names:tc:SAML:2.0:sender-vouches`

You can use the identity mapping rules to add subject confirmation information to the `STSUniversalUser`.

```
<stsuuser:Attribute name="SamlSubjectConfirmationMethod"
 type="urn:oasis:names:tc:SAML:2.0:assertion">
  <stsuuser:Value>urn:oasis:names:tc:SAML:2.0:cm:bearer
   </stsuuser:Value>
  <stsuuser:Value>urn:oasis:names:tc:SAML:2.0:cm:holder-of-key
   </stsuuser:Value>
 </stsuuser:Attribute>
```

Another way to add subject confirmation information is by using configuration properties. See the topic on "SAML 2.0 module properties" on page 153.

Note: The values set in the identity mapping rule take precedence over the settings in the configuration.

For the `SubjectConfirmationMethod` to be issued correctly, the client must sign the `RequestSecurityToken` request and include a `KeyInfo` used for the SCM when sending the `RequestSecurityToken`. To use the holder-of-key capability, the JavaScript mapping rules must be updated to insert the attribute into the STSUU.

For example:

```
<stsuuser:AttributeList>
 <stsuuser:Attribute name="SamlSubjectConfirmationMethod"
   type="urn:oasis:names:tc:SAML:2.0:assertion">
  <stsuuser:Value>urn:oasis:names:tc:SAML:2.0:cm:holder-of-key
  </stsuuser:Value>
 </stsuuser:Attribute>
</stsuuser:AttributeList>
```

SAML 1.1 module

The SAML 1.1 module validates and issues SAML 1.1 tokens.

The SAML 1.1 module is called `Saml11STSTokenModule`.

Security Assertion Markup Language 1.1 (SAML 1.1) is a version of the SAML standard for exchanging authentication and authorization data between security domains. SAML 1.1 enables web-based authentication and authorization scenarios including cross-domain single sign-on (SSO), which helps reduce the administrative overhead of distributing multiple authentication tokens to the user.

Scenarios

- Single sign-on federations
- Custom trust chains

Supported modes

- Validate
- Issue
- Exchange

Configuration properties

Validate mode

Enable one-time assertion use enforcement

Specifies whether to use the assertion or token only once.

Enable signature validation

Enables or disables validation of signatures in the token module.

Select a validation key

Specifies the validation key that the partner must use.

Use the KeyInfo of the XML signature to find the X509 certificate for signature validation
Determines the appropriate certificate for signature validation. When you select this option, you must provide the subject distinguished name that matches the certificate.

RegExp
Specifies a regular expression to validate the subject distinguished name returned in the`KeyInfo`.

Use the keystore alias to find the public key for signature validation
Specifies a public key for signature validation, which is the default. Select the certificate database and label.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Create multiple attribute statements in the Universal User
Specifies whether to keep multiple attribute statements in the groups in which they were received. This option might be necessary if your custom identity mapping rules are written to operate on one or more specific groups of attribute statements.

If you do not select this check box, multiple attribute statements are arranged into a single group (AttributeList) in the STS`UniversalUser` document. The default setting of the check box is not selected. This setting is appropriate for most configurations.

Issue and Exchange mode

Name of the organization issuing the assertions
Shows a string that specifies the name of the organization (for example, a company) that issues the SAML assertions.

Amount of time before the issue date that an assertion is considered valid (seconds)
Default: `60` seconds

There is no minimum or maximum value enforced.

This field must contain a value.

Amount of time that the assertion is valid after being issued (seconds)
Default: `60` seconds

There is no minimum or maximum value enforced.

This field must contain a value.

List the attribute types to include
Specifies the types of attributes to be inserted during token creation. The attributes consist of information about the identity (user). Use && to separate attribute types. By default, all types are supported, as indicated by the asterisk (*) wildcard character.

For example, to add user-defined attribute types `type1` and `type2`, enter:

```
type1&&type2
```

Sign SAML assertions
Select if SAML assertions must be signed.

Select the key for signing assertions
Specifies the key to use when signing SAML assertions.

Certificate Database
Select the certificate database to use for validation.

Certificate Label
Select the certificate label for validation.

Select the KeyInfo elements to include
Determines what `KeyInfo` elements to include in the digital signature when signing a SAML message or assertion. Select one or more of the following elements.

X509 Subject Key Identifier
Select to include the X.509 subject key identifier with your signature. If not selected, the subject key identifier is excluded. To change the default for this element, change it in the custom properties.

Public Key
Select to include the public key with your signature. If not selected, the public key is excluded. To change the default for this element, change it in the custom properties.

X509 Subject Issuer Details
Select to include the issuer name and the certificate serial number with your signature. If not selected, the subject issuer details are excluded. To change the default for this element, change it in the custom properties.

X509 Subject Name
Select to include the X.509 subject name with your signature. If not selected, the X.509 data is excluded. To change the default for this element, change it in the custom properties.

X509 Certificate Data
Select to include the BASE64 encoded certificate data with your signature. If not selected, the X.509 data is excluded. To change the default for this element, change it in the custom properties.

Use Inclusive Namespaces
Specifies whether to use the `InclusiveNamespaces` construct, which means employing exclusive XML canonicalization for greater standardization. The default is cleared.

Note: If you do not select any of the `KeyInfo` elements, X.509 certificate data is still included in the signature by default.

Signature algorithm for signing SAML assertions
Specifies the signature algorithm to use to sign the SAML assertion.

RSA-SHA1
`http://www.w3.org/2000/09/xmldsig#rsa-sha1`

DSA-SHA1
`http://www.w3.org/2000/09/xmldsig#dsa-sha1`

RSA-SHA256
`http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`

Note: The chosen signature algorithm must match the signing key type that was set in the federation level to prevent a signature failure. For example, select DSA-SHA1 for DSA keys.

Subject confirmation method
Specifies the subject confirmation method for the assertion. You can select one confirmation method, or choose `No Subject Confirmation Method`. If you select the holder-of-key type, the default includes the X.509 Certificate Data in the `KeyInfo` for the `SubjectConfirmationMethod`. `STSUniversalUser` can provide the data for the subject confirmation method `KeyInfo`. The data can also be extracted from the signed request data.

Valid values can be:

- `No Subject Confirmation Method`
- `urn:oasis:names:tc:SAML:1.0:bearer`
- `urn:oasis:names:tc:SAML:1.0:holder-of-key`
- `urn:oasis:names:tc:SAML:1.0:sender-vouches`

You can use the identity mapping rules to add subject confirmation information to the `STSUniversalUser`.

```
<stsuuser:Attribute name="SamlSubjectConfirmationMethod"
 type="urn:oasis:names:tc:SAML:1.0:assertion">
  <stsuuser:Value>urn:oasis:names:tc:SAML:1.0:cm:bearer
   </stsuuser:Value>
  <stsuuser:Value>urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
   </stsuuser:Value>
 </stsuuser:Attribute>
```

Another way to add subject confirmation information is by using configuration properties. See the topic on "SAML 1.1 module properties" on page 160.

Note: The values set in the identity mapping rule take precedence over the settings in the configuration.

For the `SubjectConfirmationMethod` to be issued correctly, the client must sign the `RequestSecurityToken` request and include a `KeyInfo` used for the SCM when sending the `RequestSecurityToken`. To use the holder-of-key capability, the JavaScript mapping rules must be updated to insert the attribute into the STSUU.

For example:

```
<stsuuser:AttributeList>
 <stsuuser:Attribute name="SamlSubjectConfirmationMethod"
   type="urn:oasis:names:tc:SAML:1.0:assertion">
  <stsuuser:Value>urn:oasis:names:tc:SAML:1.0:cm:holder-of-key
  </stsuuser:Value>
 </stsuuser:Attribute>
</stsuuser:AttributeList>
```

STS Universal User module

The Security Token Service Universal User (STSUU) module acts as a pass-through module to either pass in or out an XML-based STSUniversalUser token.

The STS Universal User module is called STSUU`STSModule`.

This module is useful for testing other STS modules or for simple custom trust client applications. It provides a simple means to directly call the trust service to issue more complex token types without having to first pass in another token, and then perform a mapping operation.

The input STSUniversalUser token can contain the username, any extended attributes, and any attributes required for issuing the SAML assertion, as generated by the caller of the trust service.

No mapping step is required.

Scenario

- Custom trust chains

Supported modes

- Validate
- Issue
- Exchange

Configuration properties

None.

Security Token Service Universal User document

In order to ensure that an incoming token can be converted properly into an outgoing token that contains the content and format that is required by the partner, Security Verify Access creates an intermediate document in a generic XML format that holds identity information. This document is called the STS Universal User or STSUU. The STSUU document contains three sections:

- Principal information
- Group information
- Attribute information

To create the STSUU document, Security Verify Access uses an XML schema that specifies the structure. The schema is defined in the file stsuuser.xsd. The following code sample contains the entire contents of the secure token service universal user XML schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="urn:ibm:names:ITFIM:1.0:stsuuser"
xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser"
elementFormDefault="qualified">

 <xsd:element name="STSUniversalUser">
  <xsd:complexType>
   <xsd:sequence>
     <xsd:element name="Principal" type="stsuuser:PrincipalType"
             minOccurs="1" maxOccurs="1"/>
     <xsd:element name="GroupList" type="stsuuser:GroupListType"
             minOccurs="0" maxOccurs="1"/>
     <xsd:element name="AttributeList" type="stsuuser:AttributeListType"
             minOccurs="0" maxOccurs="1"/>
     <xsd:element name="RequestSecurityToken" type="stsuuser:RequestSecurityTokenType"
             minOccurs="0" maxOccurs="1"/>
   </xsd:sequence>
   <xsd:attribute name="version" type="xsd:string" use="required"/>
  </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="PrincipalType">
  <xsd:sequence>
   <xsd:element name="Attribute" type="stsuuser:AttributeType"
         minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="RequestSecurityTokenType">
  <xsd:sequence>
   <xsd:element name="Attribute" type="stsuuser:AttributeType"
         minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="AttributeType">
                 <xsd:sequence>
                  <xsd:element name="Value" type="xsd:string"
                        minOccurs="0" maxOccurs="unbounded"/>
                  </xsd:sequence>
                 <xsd:attribute name="name" type="xsd:string" use="required"/>
                 <xsd:attribute name="type" type="xsd:string" use="optional" />
                 <xsd:attribute name="nickname" type="xsd:string" use="optional" />
                  <xsd:attribute name="preferEncryption" type="xsd:boolean" use="optional" />

 </xsd:complexType>

 <xsd:complexType name="AttributeListType">
  <xsd:sequence>
   <xsd:element name="Attribute" type="stsuuser:AttributeType"
         minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="GroupListType">
  <xsd:sequence>
   <xsd:element name="Group" type="stsuuser:GroupType"
         minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="GroupType">
  <xsd:sequence>
   <xsd:element name="Attribute" type="stsuuser:AttributeType"
         minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required" />
  <xsd:attribute name="type" type="xsd:string" use="optional" />
 </xsd:complexType>
```

```
</xsd:schema>
```

Although the schema is used as the base for all STSUU documents, the exact information contained in any specific STSUU document is dependent on the token type for the security token that was used as input. The information required in an STSUU document after transformation by identity mapping depends on:

- The token type to be generated.
- The specific mapping rule being used for the conversion.

During token processing for a typical single sign-on configuration, two STSUUs are created. One is an input STSUU, which is created from the original input token. The other is an output STSUU, which is created after the identity mapping rules are applied.

To view the Javadoc for the STSUU:

1. Log in to the local management interface.
2. Select **System** > **File Downloads**.
3. Expand `federation > doc`, and select `ISAM-javadoc.zip`.
4. Download and decompress the compressed file. View the API for `com.tivoli.am.fim.trustserver.sts.user`.

Username token module

The Username token STS module validates and issues `UsernameToken` elements.

The Username token STS module is called `UsernameTokenSTSModule`. The STS handles a Username token as both an incoming and outgoing token type.

There are three supported username and password validation methods from which to select.

Scenario

- Custom trust chains

Supported modes

- Validate
- Issue

Configuration properties (Validate mode)

Skip password validation
Do not perform password validation for the Username token. The default value is **`cleared`**.

User registry option
Select the type of user registry to use for validation.

Verify Access runtime
Validate the username and password according to the Verify Access runtime configuration.

Note: Complete the following steps before using this option:

1. Configure the runtime component. See Configure the runtime environment. During this process, you must specify an Verify Access user registry as your primary LDAP server.
2. Configure a federated user registry.

   Note: Client certificate authentication for federated directories is not supported for `UsernameTokenSTSModule`.

   See Managing federated directories.
3. Enable basic users. See Configuring the runtime to authenticate basic users.

LDAP bind DN
The username used to authenticate to the primary LDAP server. For example, `cn=SecurityMaster,secAuthority=Default`.

LDAP bind Password
The password used to authenticate to the primary LDAP server. For example, `admin`.

SSL Enabled
Select to enable SSL.

Certificate Database
The name of the certificate database to use for the SSL connection. For example, `embedded_ldap_keys.kdb`.

Verify Access user registry
Validate the username and password according to the configured Verify Access user registry. This method requires an LDAP server that you must define by using the local management interface. See Managing server connections.

Server Connection ID
The name of the server connection that holds the required LDAP settings to access the Security Verify Access registry. This property is required if password validation is not skipped.

Login Failures Persistent
Login failures are used with the three-strikes policy.

If this option is set to `false`, each process that uses this API stores the number of login failures in memory. If multiple servers are involved, the total number of login failures to trigger a strike-out might vary.

If this option is set to `true`, the strike count is stored in LDAP and shared across all servers. Therefore, an accurate count is kept in a multi-server environment.

The default is `false`.

Management Domain
The management domain of Security Verify Access. The default is `Default`.

Maximum Server Connections
The maximum number of connections that are made to the Security Verify Access registry.

The default is 16.

Generic LDAP user registry
Validate the username and password according to the configured LDAP user registry. It does not have to be an Verify Access user registry.

Server Connection ID
The name of the server connection that holds the required LDAP settings to access an LDAP user registry. This property is required if password validation is not skipped.

Maximum Server Connections
The maximum number of connections that are made to the LDAP user registry.

The default is 16.

User ID attribute
An LDAP attribute that stores the username. For example, `uid`.

LDAP Base DN
An LDAP base DN to search. For example, `o=ibm,c=us`.

User search filter
An LDAP search filter. For example, `((objectClass=person)(objectClass=ePerson))`.

Enable the time validity check, based on created time and the amount of time permitted after the issue
Specifies a required created time element on the Username token when checked. This property is enabled by default. The software compares the value of the created time element against the value that specifies the amount of time that the token is valid after it is issued.

Amount of time the token is valid after being issued
The amount of time a token is valid after it is issued. The default value is **`300`** seconds. A value of **`-1`** means that the token does not expire.

Configuration properties (Issue mode)

Include nonce in token
Includes a nonce (random bits used for obfuscating the element) in the token. When the password option **4** is specified, this value has no effect.

Include token creation time in token
Adds a time stamp to the token, indicating the creation time of the token.

Options for including password in the token
Indicates whether to include the password in the token. When the password is included, you can specify the format.

Do not include the password
Specifies that you do not want to include the password in the token.

Include the digest of the password value
Specifies that you want to include the password in the token as the digest of the password value.

Include the password in clear text
Specifies that you want to include the password in the token as clear text.

PassTicket module

The PassTicket token STS module validates and issues Resource Access Control Facility (RACF®) `PassTicket` tokens.

The PassTicket module is called `PassTicketSTSModule`. PassTicket tokens extend the structure of `Username` tokens by adding a generated PassTicket.

Scenario

- Custom trust chains

Supported modes

- Validate
- Issue
- Exchange

Configuration properties for Validate mode

Amount of time the token remains valid (seconds)
An integer value that indicates the amount of time, in seconds, that the token remains valid.

Default value is `300`.

The special value `-1` means that the token does not expire.

Hexadecimal key used to validate a PassTicket token

A key value that consists of exactly 16 hexadecimal digits, which are used to validate a valid PassTicket.

Note: Leave as ******** if you are editing the property, and the key does not need to be changed.

The name of the application used to generate the unique PassTicket

The name of the application that was used to generate the unique PassTicket. This property must be an eight character user ID. The characters must be alphanumeric. For example, `GS1SGRAM`.

Dynamic application names are supported. You can override the configured application name by supplying an application name in the SOAP request. When the module is in `Validate` mode, the application name to be used is determined as follows:

1. If an application name is supplied in `wst:Claims`, use it.
2. If an application name is not supplied in `wst:Claims`, use the name that is configured in the module.

Enable signature validation
Specifies whether to enable validation of signatures in the token module. Default is false.

Certificate database
Specifies the keystore that contains the key or certificate for validating the signatures in the PassTicket token. Required only when `Enable signature validation` is selected.

Certificate label
Specifies the certificate in the specified keystore for validating the signatures in the PassTicket token. Required only when `Enable signature validation` is selected.

Configuration properties for Issue mode and Exchange mode

Include a nonce in the PassTicket token
Specifies whether to include a nonce (random bits used for obfuscating the element) in the PassTicket token.

Add creation timestamp in the PassTicket token
Specifies whether to add a time stamp to the PassTicket token, indicating the creation time of the token.

Hexadecimal key used to generate a PassTicket token

A key value that consists of exactly 16 hexadecimal digits, which are used to generate a valid PassTicket.

Note: Leave as ******** if you are editing the property, and the key does not need to be changed.

The name of the application used to generate the unique PassTicket

The name of the application that was used to generate the unique PassTicket. Must be an eight character user ID. The characters must be alphanumeric. For example, `GS1SGRAM`.

Dynamic application names are supported. You can override the application name by supplying an application name in the SOAP request. When the module is in `Issue` mode, the application name to use is determined in the following order:

1. If an application name is supplied in `ContextAttributes`, use it.
2. If an application name is not supplied in `ContextAttributes`, but an application name is supplied in `wst:Claims`, use the `wst:Claims` name.
3. If an application name is not supplied in either `ContextAttributes` or `wst:Claims`, use the name that is configured in the module.

Enable signing of the PassTicket token
Specifies whether to enable the signing of the PassTicket token module.

Default is false.

Certificate database
Specifies the keystore that contains the key or certificate for signing the PassTicket token. Required only when `Enable signing of the PassTicket token` is selected.

Certificate label
Specifies the certificate in the specified keystore for signing the PassTicket token. Required only when `Enable signing of the PassTicket token` is selected.

JSON Web Token (JWT)

A JWT is a set of JSON claims that are signed, encrypted, or both, and are encoded into a web safe form. This set of claims might or might not include some well-known claims that are defined by the RFC.

The methods of encrypting and signing and the support for key exchange and algorithms are defined in RFCs 7515, 7516, 7517, and 7518. These RFCs cover signing, encryption, key sets, and algorithms. RFC 7519 covers JWT.

A JWT contains three Base64 encoded strings that are separated by dots (".").

For a signed JWT, these parts are:

- JWT Header - JSON
- JWT claims - JSON
- Signature - Binary data

All of these parts are Base64 URL encoded. An example JWT is shown in the following example:

```
eyJhbGciOiJIUzI1NiJ9.eyJhdWQiOiJ5b3UiLCJpc3MiOiJtZSIsInN1YiI6InRvZ
GF5IiwiZGF5IjoibW9uZGF5In0.6f14Ub6WuEuMMSa_6hkXfj5kpVAI9tkmP5vcbX1
qH3Y
```

This JWT is signed by using the algorithm HS256 and the shared key of "secret".

You can use http://jwt.io to create and validate simple signed JWTs.

JWT support

IBM Security Verify Access supports JWT by using STS to expose a JWT module.

This module can be run in the following two modes:

Validate
Consume a JWT.

Issue
Create a JWT.

Both modes support signing, encryption, and some basic validation or population of claims.

Security Verify Access supports consuming a nested JWT using the header claim `"cty":"JWT"`. However, this support applies only when the JWT is both signed and encrypted, per RFC 7519 section 11.2.

The JWT module supports the following JSON Web Algorithms.

Table 84. Signing algorithms

Algorithm	Uses symmetric key	Uses certificates	Required key size	Suggested key size
HS256	Yes	No		256 bits
RS256	No	Yes	At least 2048 bits	
ES256	No	Yes	256 bits	
HS384	Yes	No		384 bits
RS384	No	Yes	At least 2048 bits	
ES384	No	Yes	384 bits	
HS512	Yes	No		512 bits
RS512	No	Yes	At least 2048 bits	

Table 84. Signing algorithms (continued)

Algorithm	Uses symmetric key	Uses certificates	Required key size	Suggested key size
ES512	No	Yes	512 bits	

Note: A required key size indicates that an error occurs if this value is not supplied. A suggested key size indicates the minimum value to achieve a reasonable level of security.

Table 85. Encryption key agreement

Algorithm	Uses symmetric key	Uses certificates	Required key size
RSA1_5	No	Yes	At least 2048 bits
RSA-OAEP	No	Yes	At least 2048 bits
RSA-OAEP-256	No	Yes	At least 2048 bits
A128KW	Yes	No	128 bits
A192KW	Yes	No	192 bits
A256KW	Yes	No	256 bits
A128GCMKW	Yes	No	128 bits
A192GCMKW	Yes	No	192 bits
A256GCMKW	Yes	No	256 bits
dir	Yes	No	The key size that is required by the encryption algorithm (one of 128, 192, or 256 bits)
ECDH-ES	No	Yes	
ECDH-ES+A128KW	No	Yes	
ECDH-ES+A192KW	No	Yes	
ECDH-ES+A256KW	No	Yes	

Note: A required key size indicates that an error occurs if this value is not supplied.

Table 86. Content encryption algorithms

Algorithm	Uses symmetric key	Required key size
A128GCM	Yes	128 bits
A192GCM	Yes	192 bits
A256GCM	Yes	256 bits
A128-CBC-HS256	Yes	256 bits
A192-CBC-HS384	Yes	384 bits
A256CBC-HS512	Yes	512 bits

Note:

- A required key size indicates that an error occurs if this value is not supplied.

- The Content Encryption Key (CEK) is generated in most cases. When the encryption key algorithm is "dir", you must know the required key size of the CEK. Because the mode "dir" uses the provided key as the CEK.

The size of each character in the "symmetricKey" field is 8 bits. For a 128-bit key, you need to provide a 16 character key.

The previously listed algorithms are from the JWA RFC(7518) https://tools.ietf.org/html/rfc7518.

Validate mode

In validate mode, the JWT Module consumes a binary security token, which has the attribute type "`urn:com:ibm:JWT`".

When the module consumes a JWT, the following operations are performed:

1. The keys are resolved.
2. The JWT is decrypted if it was encrypted.
3. The JWT signature is verified if it was signed. If the JWT was encrypted, this step is performed on the payload of the decrypted JWT.
4. The claims are validated.
5. The STSUU is populated.

If the JWT is successfully decrypted and validated, then the STSUU attributes will contain the claims and context attributes. The claims will have the attribute type `urn:com:ibm:JWT:claim`. The context attributes will have one of the following values:

- `urn:com:ibm:JWT:header`
- `urn:com:ibm:JWT:outer_header`
- `urn:com:ibm:JWT:signature`

The following table shows the configuration properties.

Table 87. Configuration properties and usage in validate mode

Configuration property	**Description**	**Can be provided via WS-Trust claims**
signing.alg	The algorithm with which the JWT is signed.	FALSE
signing.symmetricKey	The symmetric key that is used to perform signature validation.	TRUE
signing.db	The keystore from which the certificate is sourced.	TRUE
signing.cert	The certificate label from which the public keys are sourced.	TRUE
signing.jwksUri	The JWKS URI from which the public key is retrieved.	TRUE
encryption.alg	The algorithm that is used by the JWT for key management.	FALSE
encryption.enc	The algorithm that is used by the JWT for content encryption.	FALSE
encryption.symmetricKey	The symmetric key that is used for key management.	TRUE

Table 87. Configuration properties and usage in validate mode (continued)

Configuration property	Description	Can be provided via WS-Trust claims
encryption.db	The keystore from which the private key is sourced.	TRUE
encryption.cert	The label of the certificate that contains the private key to use for decrypting the encryption key.	TRUE
iss	The Java regular expression that matches the "iss" (issuer) claim. This value is optional.	FALSE
aud	The Java regular expression that matches the "aud" (audience) claim. This value is optional.	FALSE
sub	The Java regular expression that matches the "sub" (subject) claim. This value is optional.	FALSE
validateExp	Whether the exp claim in the JWT is checked. This check requires that the "exp" (expiration time) claim be set to a time in the future.	FALSE
validateNbf	Whether the "nbf" (not before) claim in the JWT is checked. This check requires that the nbf claim be set to a time in the past.	FALSE
validateSkew	The skew to offset time checks with.	FALSE

When the module runs in validate mode, it converts the JWT into a populated STSUU. The following examples show some sample input JWT and the corresponding output STSUU.

Input example

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:rst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">

  <SOAP-ENV:Body>
    <!--  <rst:RequestSecurityTokenCollection>-->
      <rst:RequestSecurityToken>
        <wsp:AppliesTo>
          <wsa:EndpointReference>
            <wsa:Address>validate</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
        <wst:Issuer>
          <wsa:Address>validate</wsa:Address>
        </wst:Issuer>
        <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02
        /trust/Validate</wst:RequestType>
        <wst:Claims><signing.alg>HS256</signing.alg><signing.
        symmetricKey>superSecret</signing.symmetricKey></wst:Claims>
        <wst:Base>
        <wss:BinarySecurityToken xmlns:wss="http://docs.oasis-open.org
        /wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" wss:
        EncodingType="http://ibm.com/2004/01/itfim/base64encode" wss:
        ValueType="urn:com:ibm:JWT">eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoi
        am9obiIsInRpdGxlIjoiTXIiLCJleHAiOjE0NjA0MzkxNzN9.BNkZM38PygNYb
        PzGSsd1Za8HmgUkn0aT0ImaJmBmKtU</wss:BinarySecurityToken>

        </wst:Base>
      </rst:RequestSecurityToken>
      <!--</rst:RequestSecurityTokenCollection>-->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output example

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
  <soap:Body>
    <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://
    docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestSecurityTokenResponse xmlns:wsu="http://docs.oasis-open.org
      /wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
      wsu:Id="uuid8f53fcc-0154-10f4-bfcd-ebb7b0604011">
        <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:
        wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
          <wsa:EndpointReference>
            <wsa:Address>validate</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
        <wst:RequestedSecurityToken>
          <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
            <stsuuser:Principal/>
            <stsuuser:AttributeList>
              <stsuuser:Attribute name="title" type="urn:com:ibm:JWT:claim">
                <stsuuser:Value>Mr</stsuuser:Value>
              </stsuuser:Attribute>
              <stsuuser:Attribute name="exp" type="urn:com:ibm:JWT:claim">
                <stsuuser:Value>1460439173</stsuuser:Value>
              </stsuuser:Attribute>
              <stsuuser:Attribute name="name" type="urn:com:ibm:JWT:claim">
                <stsuuser:Value>john</stsuuser:Value>
              </stsuuser:Attribute>
            </stsuuser:AttributeList>
            <stsuuser:RequestSecurityToken/>
            <stsuuser:ContextAttributes>
              <stsuuser:Attribute name="header" type="urn:com:ibm:JWT">
                <stsuuser:Value>{"alg":"HS256"}</stsuuser:Value>
              </stsuuser:Attribute>
              <stsuuser:Attribute name="signature" type="urn:com:ibm:JWT">
                <stsuuser:Value>BNkZM38PygNYbPzGSsd1Za8HmgUkn0aT0ImaJmBmKtU
                </stsuuser:Value>
              </stsuuser:Attribute>
            </stsuuser:ContextAttributes>
            <stsuuser:AdditionalAttributeStatement/>
          </stsuuser:STSUniversalUser>
        </wst:RequestedSecurityToken>
        <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate
        </wst:RequestType>
        <wst:Status>
          <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status
          /valid</wst:Code>
        </wst:Status>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </soap:Body>
</soap:Envelope>
```

Issue mode

In issue mode, the JWT Module creates a binary security token, which has the attribute type `urn:com:ibm:JWT`.

When the module creates a JWT, the following operations are performed:

1. The keys are resolved.

2. The claims are populated from the STSUU.
3. The static claims are populated, if they were configured and are not already set from the STSUU.
4. The JWT is signed if signing is set.
5. The JWT is encrypted. If it is signed, the signed JWT will be encrypted and the claim `“cty”:”jwt”` will be added to the header.
6. The binary security token is issued.

The following table shows the configuration properties.

Table 88. Configuration properties and usage in issue mode

Configuration property	**Description**	**Can be provided by STSUU Context Attributes**
signing.alg	The algorithm with which the JWT is signed.	TRUE
signing.symmetricKey	The symmetric key that is used to perform signature validation.	TRUE
signing.db	The keystore from which the certificate is sourced.	TRUE
signing.cert	The certificate label from which the public keys are sourced.	TRUE
signing.kid	The Key ID that is used for signing.	TRUE
encryption.jwksUri	The JWKS URI that is used for encryption.	TRUE
encryption.kid	The Key ID that is used for encryption.	TRUE
encryption.alg	The algorithm that is used by the JWT for key management.	TRUE
encryption.enc	The algorithm that is used by the JWT for content encryption.	TRUE
encryption.symmetricKey	The symmetric key that is used for key management.	TRUE
encryption.db	The keystore from which the private key is sourced.	TRUE
encryption.cert	The label of the certificate that contains the private key to use for decrypting the encryption key.	TRUE
includeIat	A Boolean value that indicates whether the "iat" (issued at) claim is generated and included in the JWT. This value does not override an existing "iat" value if it is already present.	FALSE
iss	The static value with which the "iss" (issuer) claim is populated.	FALSE

Table 88. Configuration properties and usage in issue mode (continued)

Configuration property	Description	Can be provided by STSUU Context Attributes
aud	The static value with which the "aud" (audience) claim is populated.	FALSE
sub	The static value with which the "sub" (subject) claim is populated.	FALSE
jti	JWT ID, which is a unique identifier for the JWT. A value of 0 disables the claim.	FALSE
exp	Offset for the "exp" (expiration time) claim. A value of 0 disables the claim.	FALSE
nbf	Offset for the "nbf" (not before) claim. A value of -1 disables the claim.	FALSE

When the module runs in issue mode, it converts the STSUU into a JWT. The following examples show some sample input STSUU and the corresponding output JWT.

To add custom claims to a JWT header, add a custom context attribute with the type `"urn:ibm:JWT:header:claim"`. This type is not case sensitive.

A snippet of an example attribute in XML is shown as follows:

```
<stsuuser:ContextAttributes>...
      <stsuuser:Attribute name="typ" type="urn:ibm:JWT:header:claim">
         <stsuuser:Value>JWT</stsuuser:Value>
      </stsuuser:Attribute>...</stsuuser:ContextAttributes>
```

Input example

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:rst="http://docs.oasis-open.org/ws-sx/ws-trust/200512"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
  <SOAP-ENV:Body>
      <rst:RequestSecurityToken>
        <wsp:AppliesTo>
          <wsa:EndpointReference>
            <wsa:Address>issue</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
        <wst:Issuer>
          <wsa:Address>issue</wsa:Address>
        </wst:Issuer>
        <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate
        </wst:RequestType>
        <wst:Base>
          <stsuuser:STSUniversalUser xmlns:stsuuser="urn:ibm:names:ITFIM:1.0:stsuuser">
            <stsuuser:Principal/>
            <stsuuser:AttributeList>
              <stsuuser:Attribute name="name" type="urn:ibm:jwt:claim">
                <stsuuser:Value>john</stsuuser:Value>
              </stsuuser:Attribute>
              <stsuuser:Attribute name="title" type="urn:ibm:jwt:claim">
                <stsuuser:Value>Mr</stsuuser:Value>
              </stsuuser:Attribute>
            </stsuuser:AttributeList>
            <stsuuser:ContextAttributes>

     <!-- specify a HS256 JWT, with the key "superSecret" -->
              <stsuuser:Attribute name="signing.symmetricKey" type="">
                <stsuuser:Value>superSecret</stsuuser:Value>
              </stsuuser:Attribute>
              <stsuuser:Attribute name="signing.alg" type="">
                <stsuuser:Value>HS256</stsuuser:Value>
              </stsuuser:Attribute>
            </stsuuser:ContextAttributes>
            <stsuuser:AdditionalAttributeStatement id=""/>
          </stsuuser:STSUniversalUser>
        </wst:Base>

      </rst:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Output example

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
  <soap:Body>
    <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://
    docs.oasis-open.org/ws-sx/ws-trust/200512">
      <wst:RequestSecurityTokenResponse xmlns:wsu="http://docs.oasis-open.org/
      wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" wsu:Id=
      "uuid8f2887f-0154-1671-a234-ebb7b0604011">
        <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing" xmlns:
        wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
          <wsa:EndpointReference>
            <wsa:Address>issue</wsa:Address>
          </wsa:EndpointReference>
        </wsp:AppliesTo>
        <wst:RequestedSecurityToken>
          <wss:BinarySecurityToken xmlns:wss="http://docs.oasis-open.org/wss
          /2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" wss:EncodingType=
          "http://ibm.com/2004/01/itfim/base64encode" wss:ValueType="urn:com:ibm:JWT">
          eyJhbGciOiJIUzI1NiJ9.eyJuYW1lIjoiam9obiIsInRpdGxlIjoiTXIiLCJleHAiOjE0NjA0Mz
          kxNzN9.BNkZM38PygNYbPzGSsd1Za8HmgUkn0aT0ImaJmBmKtU</wss:BinarySecurityToken>
        </wst:RequestedSecurityToken>
        <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate
        </wst:RequestType>
        <wst:Status>
          <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid
          </wst:Code>
        </wst:Status>
      </wst:RequestSecurityTokenResponse>
    </wst:RequestSecurityTokenResponseCollection>
  </soap:Body>
</soap:Envelope>
```

Pre populating the JWT JSON

In some instances, a more complex and custom JSON format for the JWT might be necessary. This can be achieved by providing the context attribute “`claim_json`”. The value of this attribute will be parsed and used when initializing the JSON that will be the claims for the JWT. Any attributes that are present will be added to the JWT.

Attribute example:

```

<stsuuser:ContextAttributes>
...
   <stsuuser:Attribute name="claim_json" type="">
      <stsuuser:Value>
      {
      “customObjectAttribute”  : {},
      “customBooleanAttribute” : true,
      “customIntegerAttribute” : 1
      }
      </stsuuser:Value>
   </stsuuser:Attribute>
...
```

```
</stsuuser:ContextAttributes>
```

Kerberos Module

The Kerberos module is called the KerberosSTSModule.

Validates Kerberos security tokens with a token type of `http://docs.oasis-open.org/wss/oasis-wss-kerberos-token-profile-1.1#GSS_Kerberosv5_AP_REQ`.

Supported mode
`Validate`

Configuration properties
`Validate mode`

Kerberos keytab file
Specify one of the available imported Kerberos service keytab file for the Kerberos service identified by the Kerberos security token. Follow the steps below to import a keytab file:

1. From the top menu, select **Federation** > **Global Keys** > **Kerberos Keytab file**.
2. Click **Import** to upload a keytab file.

This field is required.

Service principal name for the Kerberos
Specify the principal name of the Kerberos service in the form of `<service name>/<fully qualified hostname>@<realmname>`. For example, `HTTP/WIN-JCCFTF7M7EI.kerb.com@KERB.COM`.

This field is required.

Note: If you want to overwrite the default Kerberos configuration information, navigate to **Web** > **Kerberos Configuration** > **Defaults**.

Kerberos Keytab File

Use the Keyfiles tab on the Kerberos Configuration management page in the LMI to manage these settings.

About this task

The Keyfiles tab contains settings for the keytab files that are used for Kerberos authentication. You can import, combine, and delete keytab files. You can also test authentication with a Kerberos principal name and keytab file.

Procedure

1. From the top menu, select **Federation** > **Global Keys** > **Kerberos Keytab File**.
   The current Kerberos configuration is displayed.

2. On the Keyfiles tab, take actions as needed.
   - Import a keytab file
     a. Click **Import**.
     b. In the **Import Keytab File** window, click **Browse**.
     c. Select the keytab file to be imported and then click **Open**.
     d. Click **Import**.
   - Delete a keytab file
     a. Select the file to delete from the table.
     b. Click **Delete**.
     c. In the **Confirm Action** window, click **Yes**.
   - Combine keytab files
     a. Select the keytab files to be combined from the table.
     b. Click **Combine**.
     c. In the **Combine Keytab Files** window, enter the name for the combined file in the **New Resource Name** field.
     d. Click **Save**.
   - Verify authentication with a keytab file
     a. Select the keytab file to test from the table.
     b. Click **Test**.
     c. In the **Test Keytab Authentication** window, provide the value of the Kerberos principal in the **Username** field.
     d. Click **Test**.

X.509 module

The X.509 module is called X509STSModule.

Validates X.509 security tokens with a token type of:

```
http://docs.oasis-open.org/wss/2004/01/oasis-200401-
  wss-x509-token-profile-1.0#X509
http://docs.oasis-open.org/wss/2004/01/oasis-200401-
  wss-x509-token-profile-1.0#X509v3
http://docs.oasis-open.org/wss/2004/01/oasis-200401-
  wss-x509-token-profile-1.0#X509PKIPathv1
```

The module uses the IBM Security Verify Access KESS to validate the X.509 certificate path.

Deployment scenarios for this module type

- Custom trust chains

Supported modes

- Validate

Configuration properties

Enable X.509 certificate validation
: Specifies whether validation of X.509 certificates must be enforced. By default, this check box is selected. When this box is cleared, the certificate is not validated. This option can be used in deployments where the certificate has already been validated by another entity.

X.509 default value type
: If an X.509 `BinarySecurityToken` does not have the `ValueType` attribute specified, this configuration value is used as the default `ValueType`.

Include Subject DN
If enabled, the X.509 Subject Distinguished Name is added to the `STSUniversalUser AttributeList`.

Include Issuer DN
If enabled, the X.509 Issuer distinguished name is added to the `STSUniversalUser AttributeList`.

Include Not Before
If enabled, the X.509 `NotBefore` date is added to the `STSUniversalUser AttributeList`. This date indicates the earliest date from which the X.509 is valid.

Include Not After
If enabled, the X.509 `NotAfter` date is added to the `STSUniversalUser AttributeList`. This date indicates the latest date for which the X.509 is valid.

Include Serial Number
If enabled, the X.509 serial number is added to the `STSUniversalUser AttributeList`.

Include Type
If enabled, the X.509 type is added to the `STSUniversalUser AttributeList`.

Include Version
If enabled, the X.509 version is added to the `STSUniversalUser AttributeList`.

Include Basic Constraints
If enabled, the X.509 Basic Constraints are added to the `STSUniversalUser AttributeList`.

Please enter a list of Object Identifiers to read from the certificate
Use this text area to add custom Object Identifiers to the `STSUniversalUser AttributeList`. Put each unique OID on a new line in the text area. Each value is a hexadecimal representation of the octet string.

Token module properties

Configure token modules so that it contains the appropriate values for your environment.

Attribute Mapping module properties

You can define Attribute Mapping module self or partner properties.

Table 89. Attribute Mapping module properties

Appliance property	Self or Partner	Mode	Description
`attribute.mappings`	PARTNER, SELF	Map	Attribute Mapping in the format: *`attributeName_attributeSourceID`*.

Default Mapping module properties

You can define Default Mapping module self or partner properties.

Table 90. Default Mapping module properties

Appliance property	Self or Partner	Mode	Description
`map.rule.reference.ids`	PARTNER, SELF	Map	Specifies the ID of the JavaScript file containing the identity mapping rule.

HTTP Callout module properties

You can define HTTP Callout module self or partner properties.

Table 91. HTTP Callout module properties

Appliance property	Self or Partner	Mode	Description
uri	PARTNER, SELF	Map	Specifies the endpoint address of the web service.
authType	PARTNER, SELF	Map	Specifies the client authentication type: **NONE** Specifies no authentication. **CERTIFICATE** Specifies client certificate authentication. If selected, set the following client keystore parameters: • **clientKeyStore** • **clientKeyAlias** **BASIC** Specifies basic authentication. If selected, set the basic authentication username and password parameters: • **basicAuthUsername** • **basicAuthPassword**
sslKeyStore	PARTNER, SELF	Map	Specifies the server certificate information. If the **uri** parameter is an HTTPS endpoint, then set this parameter to point to the truststore that contains the HTTPS certificate of the endpoint.
clientKeyStore	PARTNER, SELF	Map	Defines the name of the client certificate store. It is required if the **authType** parameter is set to `CERTFICATE`.
clientKeyAlias	PARTNER, SELF	Map	Defines the alias of the client certificate. It is required if the **authType** parameter is set to `CERTFICATE`.
basicAuthUsername	PARTNER, SELF	Map	Defines the basic authentication username. It is required if the **authType** parameter is set to `BASIC`.
basicAuthPassword	PARTNER, SELF	Map	Defines the plain text basic authentication password. It is required if the **authType** parameter is set to `BASIC`.

Table 91. HTTP Callout module properties (continued)

Appliance property	Self or Partner	Mode	Description
messageFormat	PARTNER, SELF	Map	Defines the message format. Supports XML or WSTrust as values.
appliesTo	PARTNER, SELF	Map	If the **messageFormat** parameter is set to WSTrust, then set this parameter to the WSTrust applies-to address. This value is typically formatted as a URL.
issuerUri	PARTNER, SELF	Map	If the **messageFormat** parameter is set to WSTrust, then set this parameter to the WSTrust issuer address. This value is typically formatted as a URL.

IVCred module properties

You can define Security Verify Access IVCred token module self or partner properties.

Table 92. IVCred module properties

Appliance Property	Self or Partner	Mode	Description
ivcred.attribute.types	SELF	Issue	Specifies the attribute type to include in the assertion. Enter one attribute type, or use an asterisk (*) for all types. The default is an asterisk (*). This property is required.
ivcred.sign.keystore.alias.db	SELF	Issue	Specifies the name of the keystore for the signing key. For example, use DefaultKeyStore. This property is required if **ivcred.add.signatures=true.**
ivcred.sign.keystore.alias.cert	SELF	Issue	Specifies the name of the signing key. For example, use testkey. This property is required if **ivcred.add.signatures=true.**
ivcred.add.signatures	SELF	Issue	Specifies that signatures must be added to tokens. Set to true to add signatures to tokens. Set to false to exclude signatures in tokens. This property is optional.

Table 92. IVCred module properties (continued)

Appliance Property	Self or Partner	Mode	Description
`ivcred.signing. IncludeX509SubjectKeyIdentifier`	SELF	Issue	Specifies whether to include this attribute. Set to `true` to include the X509 Subject Key Identifier of the signing certificate. Set to `false` to exclude the X509 Subject Key Identifier. This is the default. Required if `ivcred.add.signatures=true`.
`ivcred.signing.IncludePublicKey`	SELF	Issue	Specifies whether to include the KeyInfo element, Public Key. Set to `true` to include the Public Key. Set to `false` to exclude the Public Key. This is the default. Required if **`ivcred.add.signatures=true`**.
`ivcred.signing. IncludeX509IssuerDetails`	SELF	Issue	Specifies whether to include the Key Info element, X509 Issuer Details. Set to `true` to include the X509 Issuer Details. Set to `false` to exclude the X509 Issuer Details. This is the default. Required if **`ivcred.add.signatures=true`**.
`ivcred.IncludeX509SubjectName`	SELF	Issue	Specifies whether to include the Key Info element, X509 Subject Name. Set to `true` to include the X509 Subject Name. Set to `false` to exclude the X509 Subject Name. This is the default. Required if **`ivcred.add.signatures=true`**.

Table 92. IVCred module properties (continued)

Appliance Property	Self or Partner	Mode	Description
ivcred.IncludeX509CertificateData	SELF	Issue	Specifies whether to include the Key Info element, X509 Certificate Data. Set to `true` to include the X509 Certificate Data. Set to `false` to exclude the X509 Certificate Data. This is the default. Required if **ivcred.add.signatures=true**.
ivcred.validate.keystore.alias.db	PARTNER	Validate	Specifies the name of the keystore for the key identifier. For example, use `DefaultKeyStore`. Required if **ivcred.verify.signatures=true**.
ivcred.validate.keystore.alias.cert	PARTNER	Validate	Specifies the name of the validation key identifier. For example, use `testkey`. Required if **ivcred.verify.signatures=true**.
ivcred.verify.signatures	PARTNER	Validate	Specifies whether the signatures are verified. Set to `true` to verify signatures. Set to `false` for no signature verification. The default is `false`. This property is optional.

LTPA module properties

You can define LTPA token module self or partner properties.

Table 93. LTPA module properties

Appliance property	Self or Partner	Mode	Description
ltpa.self.filename	SELF	Issue	Specifies the LTPA file to use. This property is required.
ltpa.self.password	SELF	Issue	Specifies the password that was used to protect the keys. It must be the same password that was used when the keys were created. This property is required.
ltpa.self.expiration	SELF	Issue	Specifies the expiration, in minutes, set on created tokens. The default is 120.

Table 93. LTPA module properties (continued)

Appliance property	Self or Partner	Mode	Description
`ltpa.self.extattr`	SELF	Issue	Specifies the attribute type to add to a version 2 token. Enter one attribute type, or use an asterisk (*) for all types.
`ltpa.self.realm`	SELF	Issue	Specifies the realm used to create the user name in the token.
`ltpa.self.usefips`	SELF	Issue	Specifies whether FIPS mode should be used for incoming tokens. The default is `false`.
`ltpa.self.version`	SELF	Issue	Specifies the version of token to be created. This property is required.
`ltpa.partner.filename`	PARTNER	Validate	Specifies the name of the previously imported LTPA file. This property is optional.
`ltpa.partner.password`	PARTNER	Validate	Specifies the password that was used to protect the keys created by the partner. It must be the same password that was used when the keys were created by the partner. This property is optional.
`ltpa.partner.usefips`	PARTNER	Validate	Specifies whether FIPS mode should be used for incoming tokens.

SAML 2.0 module properties

You can define SAML 2.0 token module self or partner properties.

Table 94. SAML 2.0 module properties

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.2.0.assertion.replay.validation`	SELF	Validate	Specifies whether to enable one-time assertion use enforcement. Set to `true` to enable one-time use enforcement. Set to `false` if you do not want to enforce one-time assertion use. **Note:** If the assertion to be validated has `<saml:OneTimeUse></saml:OneTimeUse>` in the assertion conditions, then the one-time assertion use is enforced even though the property is disabled.

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.2.0. assertion.verify.signatures	PARTNER	Validate	Specifies whether to enable signature validation. Set to `true` to enable validation. Set to `false` if you do not want validation enabled.
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.use.keyinfo	PARTNER	Validate	Specifies whether to use the `KeyInfo` of the XML signature to find the X509 certificate for signature validation. Set to `true` to use this method. Then, define the `com.tivoli.am.fim.sts.saml.2.0.Valida` property. Set to `false`, otherwise.
com.tivoli.am.fim.sts.saml.2.0. assertion.keystore.alias	PARTNER	Validate	Specifies whether to use the keystore alias to find the public key for signature validation. Set to `true` to use this method. Then, define the `com.tivoli.am.fim.sts.saml.2.0.Valida` and `com.tivoli.am.fim.sts.saml.2.0.Valida` properties. Set to `false`, otherwise.
com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier	PARTNER	Validate	Specifies a regular expression to validate the subject distinguished name returned in the `KeyInfo`, if **com.tivoli.am.fim.sts.saml.2.0. assertion.signature.use.keyinfo** is set to `true`. You can either specify this property **or** specify both of the following properties: • **com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier.db** • **com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier.cert** If you specify all of these properties, the keystore alias format overwrites the **com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier** property.

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier.db	PARTNER	Validate	Specifies the name of the certificate database to use for validation, if com.tivoli.am.fim.sts.saml.2.0.assert keystore.alias is set to true.
com.tivoli.am.fim.sts.saml.2.0. ValidateKeyIdentifier.cert	PARTNER	Validate	Specifies the name of the certificate label for validation, if com.tivoli.am.fim.sts.saml.2.0.assert is set to true.
com.tivoli.am.fim.sts.saml.2.0. DecryptionKeyIdentifier.db	PARTNER	Validation	Specifies the name of the keystore for the decryption key. For example, use DefaultKeyStore.
com.tivoli.am.fim.sts.saml.2.0. DecryptionKeyIdentifier.cert	PARTNER	Validation	Specifies the name of decryption key. For example, use testkey.
com.tivoli.am.fim.sts.saml.2.0. WantMultipleAttributeStatements	PARTNER	Validate	Specifies whether to create multiple attribute statements in the Universal User. If you specify false, multiple attribute statements are arranged into a single group (AttributeList) in the STSUniversalUserdocument. This setting is appropriate for most configurations.
com.tivoli.am.fim.sts.saml.2.0. map.unknown.alias	PARTNER	Validate	Specifies whether to map unknown name identifiers to the anonymous username.

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.2.0. assertion.default.nameidformat	PARTNER	Validate	Specifies the default NameID format for assertion validation. Specify a parameter for use during validation of a SAML assertion. The parameter determines processing rules for the NameID element when one of the following conditions exists: • If there is not an explicit Format attribute included in the assertion. • If the Format attribute is: `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified`. Typically, this parameter is needed only for STS chains that process SAML assertions that do not set the Format attribute. A normal example value is `:urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress`
com.tivoli.am.fim.sts.saml.2.0. assertion.issuer	SELF	Issue, Exchange	Specifies the name of the organization that issues assertions. This is required.
com.tivoli.am.fim.sts.saml.2.0. assertion.pretime.valid	SELF	Issue, Exchange	Specifies the number of seconds that assertions are valid before its issue date. There is no minimum or maximum value enforced, but a value is required. Default: `60`
com.tivoli.am.fim.sts.saml.2.0. assertion.posttime.valid	SELF	Issue, Exchange	Specifies the number of seconds that assertions are valid after its issue date. There is no minimum or maximum value enforced, but a value is required. Default: `60`
com.tivoli.am.fim.sts.saml.2.0.assertion.signature.use.inclusive.namespaces	PARTNER	Issue, Exchange	Specifies whether to use the InclusiveNamespaces construct. This means using exclusive XML canonicalization for greater standardization. You must set this parameter without a prefix. Set to `true` or `false`. If unset, the system behaves as if it was set to `true`.

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.2.0. **assertion.attribute.types**	PARTNER	Issue, Exchange	Specifies the types of attributes to include in the assertion. The default, an asterisk (*), includes all the attribute types that are specified in the identity mapping file. To specify one or more attribute types individually, enter each attribute type. Separate multiple type values using &&.
com.tivoli.am.fim.sts.saml.2.0. assertion.sign	PARTNER	Issue, Exchange	Specifies whether SAML assertions must be signed. Set to `true` to sign assertions. Set to `false` if signing is not required.
com.tivoli.am.fim.sts.saml.2.0. SigningKeyIdentifier.db	PARTNER	Issue, Exchange	Specifies the name of the keystore where the signing key is stored. For example, use `DefaultKeyStore`.
com.tivoli.am.fim.sts.saml.2.0. signingKeyIdentifier.cert	PARTNER	Issue, Exchange	Specifies the name of the signing key identifier. For example, use `testkey`.
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.include. subject.keyid	PARTNER	Issue, Exchange	Specifies whether to include the subject key identifier with your signature. Set to `true` to include the subject key identifier. Set to `false` to exclude the subject key identifier.
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.include. public.key	PARTNER	Issue, Exchange	Specifies whether to include the public key with your signature. Set to `Yes` to include the public key. Set to No to exclude the public key.
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.include. issuer.details	PARTNER	Issue, Exchange	Specifies whether to include the issuer details with your signature. Set to `Yes` to include the issuer details. Set to No to exclude the issuer details.

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.include. subject.name	PARTNER	Issue, Exchange	Specifies whether to include the subject name with your signature. Set to Yes to include the subject name. Set to No to exclude the subject name.
com.tivoli.am.fim.sts.saml.2.0. assertion.signature.include. cert.data	PARTNER	Issue, Exchange	Specifies whether to include the certificate data with your signature. Set to Yes to include the certificate data. Set to No to exclude the certificate data. If none of the **assertion.signature.include.*** properties are set, the system behaves as if **com.tivoli.am.fim.sts.saml.2.0.assert** is set to true.
com.tivoli.am.fim.sts.saml.2.0. SignatureAlgorithm	PARTNER	Issue, Exchange	Specifies the signature algorithm to use for signing assertions. Valid values: • RSA-SHA1, set to `http://www.w3.org/2000/09/xmldsig#rsa-sha1` • DSA-SHA1, set to `http://www.w3.org/2000/09/xmldsig#dsa-sha1` • RSA-SHA256, set to `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`
com.tivoli.am.fim.sts.saml.2.0. DigestAlgorithm	PARTNER	Issue, Exchange	Specifies the digest algorithm used to sign SAML messages. Valid values: • SHA1, set to `http://www.w3.org/2000/09/xmldsig#sha1` • SHA256, set to `http://www.w3.org/2001/04/xmlenc#sha256` • SHA512, set to `http://www.w3.org/2001/04/xmlenc#sha512`

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.2.0.EncryptAssertions`	PARTNER	Issue, Exchange	Specifies whether assertions are to be encrypted. Set to `true` to encrypt. Set to `false`, if no encryption is required. .
`com.tivoli.am.fim.sts.saml.2.0.EncryptionKeyIdentifier.db`	PARTNER	Issue, Exchange	Specifies the name of the keystore where the encryption key is stored. For example, use `DefaultKeyStore`.
`com.tivoli.am.fim.sts.saml.2.0.EncryptionKeyIdentifier.cert`	PARTNER	Issue, Exchange	Specifies the name of the encryption key. For example, use `testkey`.
`com.tivoli.am.fim.sts.saml.2.0.EncryptAllAttributes`	PARTNER	Issue, Exchange	Specifies whether all `Attribute` elements within the assertions are to be encrypted. Set to `true` to encrypt. Set to `false` if no encryption is required.
`com.tivoli.am.fim.sts.saml.2.0.EncryptNameIdentifiers`	PARTNER	Issue, Exchange	Specifies whether `NameID` elements in the assertions are to be encrypted. Set to `true` to encrypt. Set to `false` if no encryption is required.
`com.tivoli.am.fim.sts.saml.2.0.BlockEncryptionAlgorithm`	PARTNER	Issue, Exchange	Specifies the block encryption algorithm. • TRIPLEDES, set to `http://www.w3.org/2001/04/xmlenc#tripledes-cbc` • AES-128, set to `http://www.w3.org/2001/04/xmlenc#aes128-cbc` • AES-192, set to `http://www.w3.org/2001/04/xmlenc#aes192-cbc` • AES-256, set to `http://www.w3.org/2001/04/xmlenc#aes256-cbc`

Table 94. SAML 2.0 module properties (continued)

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.2.0.EncryptionKeyTransportAlgorithm`	PARTNER	Issue, Exchange	Specifies the key transport algorithm used to encrypt SAML messages. Valid values are: • `RSA-v1.5`, set to `http://www.w3.org/2001/04/xmlenc#rsa-1_5` • `RSA-OAEP`, set to `http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p`
`com.tivoli.am.fim.sts.saml.2.0.assertion.SubjectConfirmationMethod`	PARTNER	Issue, Exchange	Specifies the subject confirmation method. Valid values: • `urn:oasis:names:tc:SAML:2.0:cm:bear` • `urn:oasis:names:tc:SAML:2.0:cm:hold of-key` • `urn:oasis:names:tc:SAML:2.0:cm:send vouches`

SAML 1.1 module properties

You can define SAML 1.1 token module self or partner properties.

Table 95. SAML 1.1 module properties

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.1.1.assertion.replay.validation`	SELF	Validate	Specifies whether to enable one-time assertion use enforcement. Set to `true` to enable one-time use enforcement. Set to `false` if you do not want to enforce one-time assertion use.
`com.tivoli.am.fim.sts.saml.1.1.assertion.verify.signatures`	PARTNER	Validate	Specifies whether to enable signature validation. Set to `true` to enable validation. Set to `false` if you do not want validation enabled.
`com.tivoli.am.fim.sts.saml.1.1.assertion.signature.use.keyinfo`	PARTNER	Validate	Specifies whether to use the `KeyInfo` of the XML signature to find the X.509 certificate for signature validation. Set to `true` to use this method. Then, define the **`com.tivoli.am.fim.sts.saml.1.1.ValidateKeyIdentifier.keyinfo`** property. Set to `false`, otherwise.

Table 95. SAML 1.1 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.keyinfo	PARTNER	Validate	Specifies a regular expression to validate the subject distinguished name returned in the `KeyInfo`, if **com.tivoli.am.fim.sts.saml.1.1. assertion.signature.use.keyinfo**is set to `true`. You can either specify this property **or** specify both of the following properties: • **com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.db** • **com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.cert** If you specify all of these properties, the keystore alias format overwrites the **com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.keyinfo** property.
com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.db	PARTNER	Validate	Specifies the name of the certificate database to use for validation, if **com.tivoli.am.fim.sts.saml.1.1. assertion.keystore.alias** is set to `true`.
com.tivoli.am.fim.sts.saml.1.1. ValidateKeyIdentifier.cert	PARTNER	Validate	Specifies the name of the certificate label for validation, if **com.tivoli.am.fim.sts.saml.1.1. assertion.keystore.alias** is set to `true`.
com.tivoli.am.fim.sts.saml.1.1. WantMultipleAttributeStatements	PARTNER	Validate	Specifies whether to create multiple attribute statements in the Universal User. If you specify `false`, multiple attribute statements are arranged into a single group (AttributeList) in the `STSUniversalUser`document. This setting is appropriate for most configurations.
com.tivoli.am.fim.sts.saml.1.1. assertion.issuer	SELF	Issue, Exchange	Specifies the name of the organization that issues assertions. This is required.

Table 95. SAML 1.1 module properties (continued)

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.1.1.assertion.pretime.valid`	SELF	Issue, Exchange	Specifies the number of seconds that assertions are valid before its issue date. There is no minimum or maximum value enforced, but a value is required. Default: `60`
`com.tivoli.am.fim.sts.saml.1.1.assertion.posttime.valid`	SELF	Issue, Exchange	Specifies the number of seconds that assertions are valid after its issue date. There is no minimum or maximum value enforced, but a value is required. Default: `60`
`com.tivoli.am.fim.sts.saml.1.1.assertion.signature.use.inclusive.namespaces`	PARTNER	Issue, Exchange	Specifies whether to use the InclusiveNamespaces construct. This means using exclusive XML canonicalization for greater standardization. You must set this parameter without a prefix. Set to `true` or `false`. If unset, the system behaves as if it was set to `true`.
`com.tivoli.am.fim.sts.saml.1.1.assertion.attribute.types`	PARTNER	Issue, Exchange	Specifies the types of attributes to include in the assertion. The default, an asterisk (`*`), includes all the attribute types that are specified in the identity mapping file. To specify one or more attribute types individually, enter each attribute type. Separate multiple type values using &&.
`com.tivoli.am.fim.sts.saml.1.1.assertion.sign`	PARTNER	Issue, Exchange	Specifies whether SAML assertions must be signed. Set to `true` to sign assertions. Set to `false` if signing is not required.
`com.tivoli.am.fim.sts.saml.1.1.SigningKeyIdentifier.db`	PARTNER	Issue, Exchange	Specifies the name of the keystore where the signing key is stored. For example, use `DefaultKeyStore`.
`com.tivoli.am.fim.sts.saml.1.1.signingKeyIdentifier.cert`	PARTNER	Issue, Exchange	Specifies the name of the signing key identifier. For example, use `testkey`.

Table 95. SAML 1.1 module properties (continued)

Appliance property	Self or Partner	Mode	Description
com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.subject.keyid	PARTNER	Issue, Exchange	Specifies whether to include the subject key identifier with your signature. Set to `true` to include the subject key identifier. Set to `false` to exclude the subject key identifier.
com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.public.key	PARTNER	Issue, Exchange	Specifies whether to include the public key with your signature. Set to Yes to include the public key. Set to No to exclude the public key.
com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.issuer.details	PARTNER	Issue, Exchange	Specifies whether to include the issuer details with your signature. Set to Yes to include the issuer details. Set to No to exclude the issuer details.
com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.subject.name	PARTNER	Issue, Exchange	Specifies whether to include the subject name with your signature. Set to Yes to include the subject name. Set to No to exclude the subject name.
com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.cert.data	PARTNER	Issue, Exchange	Specifies whether to include the certificate data with your signature. Set to Yes to include the certificate data. Set to No to exclude the certificate data. If none of the **assertion.signature.include.*** properties are set, the system behaves as if **com.tivoli.am.fim.sts.saml.1.1.assertion.signature.include.cert.data** is set to true.

Table 95. SAML 1.1 module properties (continued)

Appliance property	Self or Partner	Mode	Description
`com.tivoli.am.fim.sts.saml.1.1.SignatureAlgorithm`	PARTNER	Issue, Exchange	Specifies the signature algorithm to use for signing assertions. Valid values: • RSA-SHA1, set to `http://w ww.w3.org/2000/09/xmldsig#rsa-sha1` • RSA-SHA256, set to `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256` • RSA-SHA512, set to `http://www.w3.org/2001/04/xmldsig-more#rsa-sha512`
`com.tivoli.am.fim.sts.saml.1.1.assertion.SubjectConfirmationMethod`	PARTNER	Issue, Exchange	Specifies the subject confirmation method. Valid values: • `No Subject Confirmation Method` • `urn:oasis:names:tc:SAML:1.1:cm:bear` • `urn:oasis:names:tc:SAML:1.1:cm:hold of-key` • `urn:oasis:names:tc:SAML:1.1:cm:send vouches`

Username module properties

You can define Username module self or partner properties.

Table 96. Username module properties

Appliance property	Self or Partner	Mode	Description
`username.password.options`	PARTNER, SELF	Issue	Specifies the option for including the password in the token: **2** Include the digest of the password value **3** Include the password in clear text **4** Do not include the password Default value is 4.

Table 96. Username module properties (continued)

Appliance property	Self or Partner	Mode	Description
username.add.nonce	SELF	Issue	Specifies whether to include the nonce (random bits used for obfuscating the element) in the token. The default is `true`. Set to `true` to include a nonce in the token. Set to `false` to exclude the nonce. When you specify to issue no password, this value is ineffective.
username.add.timestamp	SELF	Issue	Specifies whether to include creation time, or timestamp, in the token. The default is `true`. Set to `true` to add the timestamp. Set to `false` to exclude the timestamp.
username.password.validator	SELF	Validate	Specifies the user registry option to use. Valid values are: • `ISAMRTE`, for the Verify Access runtime option • `TAMRD`, for the Verify Access user registry option • `LDAP`, for the non-Verify Access user registry option
username.skip.password.validation	SELF	Validate	Specifies whether to disable password validation. The default is `false`. Set to `true` to skip validation. Set to `false` to enable validation.
username.server.connection.id	SELF	Validate	If TAMRD is specified for **username.password.validator**, specify the server connection ID. This is the name of the previously configured server connection which holds the settings for the Verify Access LDAP registry. This property is required if password validation is not skipped.
username.tamrd.management.domain	SELF	Validate	If TAMRD is specified for **username.password.validator**, specify the Verify Access management domain. The default is `Default`.

Table 96. Username module properties (continued)

Appliance property	Self or Partner	Mode	Description
`username.tamrd.login.failures.persistent`	SELF	Validate	If TAMRD is specified for **`username.password.validator`**, specify if log in failures are persistent. The default is `false`. Set to `true` to persist the failures. Set to `false` to not persist.
`username.tamrd.maximum.server.connections`	SELF	Validate	If TAMRD is specified for **`username.password.validator`**, specify the maximum number of server connections that are allowed. The default is 16.
`username.rte.bind.dn`	SELF	Validate	If ISAMRTE is specified for **`username.password.validator`**, specify the username used to authenticate to the primary LDAP server. For example, `cn=SecurityMaster,secAuthority=Defaul`
`username.rte.bind.pwd`	SELF	Validate	If ISAMRTE is specified for **`username.password.validator`**, specify the password used to authenticate to the primary LDAP server.
`username.rte.enableSSL`	SELF	Validate	Specifies whether to enable SSL. The default is `false`. Set to `true` to enable SSL. Then, define the **`username.rte.sslTrustStore`** property. Set to `false` to disable SSL.
`username.rte.sslTrustStore`	SELF	Validate	Specifies the name of the certificate database to use for the SSL connection, if **`username.rte.enableSSL`** is set to `true`.
`username.ldap.server.connection.id`	SELF	Validate	If LDAP is specified for **`username.password.validator`**, specify the name of the server connection that holds the required LDAP settings to access the LDAP user registry. For example, `my-isam-user-registry`.
`username.ldap.maximum.server.connections`	SELF	Validate	If LDAP is specified for **`username.password.validator`**, specify the maximum number of connections to make to the LDAP user registry. For example, 16.

Table 96. Username module properties (continued)

Appliance property	Self or Partner	Mode	Description
`username.ldap.base.dn`	SELF	Validate	If LDAP is specified for **`username.password.validator`**, specify an LDAP base DN to search. For example, dn `o=ibm,c=us`.
`username.ldap.search.filter`	SELF	Validate	If LDAP is specified for **`username.password.validator`**, specify an LDAP search filter. For example, `((objectClass=ePerson)(objectClass=Person))`.
`username.ldap.user.id.attribute`	SELF	Validate	If LDAP is specified for **`username.password.validator`**, specify an LDAP attribute that stores the username. The LDAP attribute must uniquely identify a user. For example, `uid`.
`username.validate.freshness`	PARTNER	Validate	Enables the time validity check, based on created time and the amount of time permitted after the issue. The default is `true`. Set to `true` to validate freshness. Set to `false` for no validation. If this property is not set, then the value of the property **`username.freshness.limit`** is checked to see if the time validation check needs to be performed.
`username.freshness.limit`	PARTNER	Validate	Specifies, in seconds, the amount of time the Username token is valid after being issued. Default: `300` seconds A value of `-1` means that the token does not expire.

STSRequest and STSResponse access using a JavaScript mapping rule

By using the Default Mapping STS Module and a JavaScript mapping rule, you can perform identity mapping. The mapping rule can access STSRequest and STSResponse objects.

The following two implicit objects and the classes required by these two objects can be exposed (for example, Java DOM, XML classes, and so on):

- STSRequest which represents the WS-Trust request
- STSResponse, which represents the WS-Trust response

Use JavaScript code `stsrequest.getRequestSecurityToken().getBase()` to get the input security token from the WS-Trust request. This returns the input security token as an instance of the Java class org.w3c.dom.Element.

Use JavaScript code `stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken` (outputSecurityToken) to set the output security token in the WS-Trust response. The outputSecurityToken is the output security token represented as an instance of Java class org.w3c.dom.Element. By default, WS-Trust response contains only one output security token. To return additional output security tokens, you can use the following JavaScript code:

```
stsresponse.addRequestSecurityTokenResponse().setRequestedSecurityToken(outputSecurityToken)
```

The examples in the following topics show the mapping to and from a base64 encoded JSON string. They use the Default Mapping module with a JavaScript mapping rule. The JavaScript mapping rule accesses the STSRequest and STSResponse objects and performs the identity mapping.

Mapping a base64 encoded JSON string to a SAML2 token example

You can map a base64 encoded JSON string to a SAML 2 token by using a JavaScript mapping rule.

About this task

The steps show an end-to-end JSON to SAML2 mapping. "STSRequest and STSResponse access using a JavaScript mapping rule" on page 167 provides a description of this support.

Procedure

1. Create a JavaScript mapping rule by using the local management interface.
   a) Select **Federation** > **Global Settings** > **Mapping Rules**.
   b) Click **Add**.
   c) In the **Content** field, copy and paste the following code:

```
    importClass(com.tivoli.am.fim.base64.BASE64Utility);
    importClass(com.tivoli.am.fim.trustserver.sts.uuser.Attribute);

    var jwtElement = stsrequest.getRequestSecurityToken().getBase();
    var jwtText    = jwtElement.getTextContent();
    var jwtString  = new java.lang.String(BASE64Utility.decode(jwtText), "UTF-8");
    var jwt        = JSON.parse(jwtString);

    for (var name in jwt) {
      if (jwt.hasOwnProperty(name)) {
        if ("sub".equals(name)) {
          stsuu.addPrincipalAttribute(new Attribute("name",
   "urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress", jwt[name]));
        } else {
          stsuu.addAttribute(new Attribute(name,
  "urn:oasis:names:tc:SAML:2.0:attrname-format:basic", jwt[name]));
        }
      }
    }
```

   d) In the **Name** field, enter `jwt_saml`.
   e) In the **Category** field, select **SAML2_0**.
   f) Click **Save** and deploy the changes.

2. Assemble the Security Token Service (STS) template.
   a) Select **Federation** > **Manage** > **Security Token Service**.
   b) Click **Templates**.
   c) Click **Add** and name the template `JSON to SAML2`. Click **OK**.
   d) Select the `JSON to SAML2` template and add the Default Map Module in Map mode and a Default SAML 2.0 token in Issue mode.
   e) Save and deploy the changes.
3. Create an STS chain that references the mapping rule and template you created in the previous steps.
   a) Within the **Security Token Service** panel, select **Module Chains**.
   b) Click **Add** to create the module chain, with the following values:

Table 97. JSON to SAML2 module chain values

Tab: Field	**Value**
Overview: Name	JSON to SAML2
Overview: Description	base64 encoded JSON string to SAML2 conversion STS chain
Overview: Template	JSON to SAML2
Lookup: Request Type	Validate
Lookup: Applies to Address	`jwtappliesto`
Lookup: Issuer Address	`jwtissuer`
Properties: Default Map Module (JavaScript file containing the identity mapping rule	`jwt_saml`
Properties: Default SAML 2.0 Token (Name of the organization issuing the assertions)	isam
Properties: Default SAML 2.0 Token (Amount of time before the issue date that an assertion is considered valid)	60
Properties: Default SAML 2.0 Token (Amount of time that the assertion is valid after being issued)	60
Properties: Default SAML 2.0 Token (List of attribute types to include)	*

   Use the defaults for all of the fields that are not specified in the table.
   c) Save and deploy the changes.
4. Use **`curl`** to test the chain.
   a) Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">jwtissuer</wsa:Address>
```

```
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">
          <wsa:Address>jwtappliesto</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
<JWT>ewogICJlbWFpbCI6ICJqb2huLmRvZUBleGFtcGxlLmNvbSIsIAogICJmYW1pbHlfbmFtZSI6ICJkb2UiLCAK
ICAiZ2l2ZW5fbmFtZSI6ICJqb2huIiwgCiAgImlzcyI6ICJpc2FtIiwgCiAgInN1YiI6ICIwMTIzNDU2Nzg5Igp9</
JWT>
      </wst:Base>
    </ns1:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bold embedded element, **<JWT> </JWT>**, is the input to the chain. This is a Base64 encoded JSON string that contains the following data::

```
{
 "email": "john.doe@example.com",
  "family_name": "doe",
  "given_name": "john",
  "iss": "isam",
  "sub": "0123456789"
}
```

b) Save this file as `jwt.xml`.

c) Run the following **`curl`** command, where `jwt.xml` is the WS-Trust 1.2 message:

```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary
@jwt.xml https://ip-rte/TrustServer/SecurityTokenService
```

The following results are returned:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-
ENV:Header>
    <soap:Body>
        <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/
ws-sx/ws-trust/200512">
            <wst:RequestSecurityTokenResponse xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
            wsu:Id="uuidc1288a62-0153-1f8b-bf2a-b4c46f51cd03">
                <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                    <wsa:EndpointReference>
                        <wsa:Address>jwtappliesto</wsa:Address>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <wst:Lifetime xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
                    <wsu:Created>2016-03-29T06:56:13Z</wsu:Created>
                    <wsu:Expires>2016-03-29T06:57:13Z</wsu:Expires>
                </wst:Lifetime>
               <wst:RequestedSecurityToken>
                    <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ID="Assertion-
uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03"
                    IssueInstant="2016-03-29T06:56:13Z" Version="2.0">
                        <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:entity">isam</saml:Issuer>
                        <saml:Subject>
                            <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:emailAddress">
                            0123456789</saml:NameID>
                            <saml:SubjectConfirmation
 Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                                <saml:SubjectConfirmationData
 NotOnOrAfter="2016-03-29T06:57:13Z"></saml:SubjectConfirmationData>
                            </saml:SubjectConfirmation>
                        </saml:Subject>
                        <saml:Conditions NotBefore="2016-03-29T06:55:13Z"
 NotOnOrAfter="2016-03-29T06:57:13Z">
                            <saml:AudienceRestriction>
                                <saml:Audience>jwtappliesto</saml:Audience>
                            </saml:AudienceRestriction>
                        </saml:Conditions>
                        <saml:AuthnStatement AuthnInstant="2016-03-29T06:56:13Z">
                            <saml:AuthnContext>

 <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password
                                </saml:AuthnContextClassRef>
                            </saml:AuthnContext>
                        </saml:AuthnStatement>
                        <saml:AttributeStatement>
                            <saml:Attribute Name="given_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">john</
saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="email"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue
 xsi:type="xs:string">john.doe@example.com</saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="iss"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">isam</
saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="family_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">doe</
saml:AttributeValue>
                            </saml:Attribute>
                        </saml:AttributeStatement>
                    </saml:Assertion>
                </wst:RequestedSecurityToken>
```

The JSON string is mapped into the SAML assertion, as shown by the previous bold text. The attributes in the SAML2 assertion are mapped from JSON attributes.

```
<wst:RequestedAttachedReference xmlns:wss="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd">
                    <wss:SecurityTokenReference xmlns:wss11="http://docs.oasis-open.org/
wss/oasis-wss-wssecurity-secext-1.1.xsd"
                    wss11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
profile-1.1#SAMLV2.0">
                        <wss:KeyIdentifier
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                        xmlns:wss="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">
                        Assertion-uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03</
wss:KeyIdentifier>
                    </wss:SecurityTokenReference>
                </wst:RequestedAttachedReference>
                <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</
wst:RequestType>
                <wst:Status>
                    <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/
valid</wst:Code>
                </wst:Status>
            </wst:RequestSecurityTokenResponse>
        </wst:RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

Related tasks

“Mapping a SAML2 token to a base64 encoded JSON string example” on page 172

You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

Mapping a SAML2 token to a base64 encoded JSON string example

You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

About this task

The steps show an end-to-end SAML to JSON mapping. "STSRequest and STSResponse access using a JavaScript mapping rule" on page 167 provides a description of this support.

Procedure

1. Create a JavaScript mapping rule using the local management interface.
   a) Select **Federation** > **Global Settings** > **Mapping Rules**.
   b) Click **Add**.
   c) In the **Content** field, copy and paste the following code:

```
   importClass(com.tivoli.am.fim.base64.BASE64Utility);
   importClass(com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils)

   var jwt = {};

   var it = stsuu.getPrincipalAttributes();
   var jt = stsuu.getAttributes();

   while (it.hasNext()) {
     var attribute = it.next();
     var name      = new String(attribute.getName());
     var value     = new String(attribute.getValues()[0]);

     if ("name".equals(name)) {
       jwt["sub"] = value;
     } else {
       jwt[name] = value;
     }
   }

   while (jt.hasNext()) {
     var attribute = jt.next();
     var name      = new String(attribute.getName());
     var value     = new String(attribute.getValues()[0]);

     jwt[name] = value;
   }

   var document   = IDMappingExtUtils.newXMLDocument();
   var jwtString  = JSON.stringify(jwt);
   var jwtText    = document.createTextNode(BASE64Utility.encode((new
java.lang.String(jwtString)).getBytes("UTF-8")));
   var jwtElement = document.createElement("JWT");

   jwtElement.appendChild(jwtText);

   stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken(jwtElement);
```

   d) In the **Name** field, enter `saml_jwt`.
   e) In the **Category** field, select SAML2_0.
   f) Click **Save** and deploy the changes.

2. Assemble the Security Token Service (STS) template.
   a) Select **Federation** > **Manage** > **Security Token Service**.
   b) Click **Templates**.
   c) Click **Add** and name the template `SAML2 to JSON`. Click **OK**.
   d) Select the `SAML2 to JSON` template and add the Default SAML 2.0 Token in Validate mode and a Default Map Module in Map mode.
   e) Save and deploy the changes.
3. Create an STS chain that references the mapping rule and template you created in the previous steps.
   a) Within the **Security Token Service** panel, select **Module Chains**.
   b) Click **Add** to create a module chain, with the following values:

Table 98. SAML2 to JSON module chain values

Tab: Field	**Value**
Overview: Name	SAML2 to JSON
Overview: Description	SAML2 to base64 encoded JSON string conversion STS chain
Overview: Template	SAML2 to JSON
Lookup: Request Type	Validate
Lookup: Applies to Address	SAML2_AppliesTo
Lookup: Issuer Address	SAML2_Issuer
Properties: Default Map Module (JavaScript file containing the identity mapping rule	`saml_jwt`

   Use the defaults for all of the fields not in the table.

   c) Save and deploy the changes.
4. Use **`curl`** to test the chain.
   a) Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">SAML2_Issuer</wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">
          <wsa:Address>SAML2_AppliesTo</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
<wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
        ID="Assertion-uuidbcb46a39-0153-1337-8efa-fec506fb7461"
 IssueInstant="2016-03-28T10:10:53Z" Version="2.0">
          <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">isam</
saml:Issuer>
          <saml:Subject>
            <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:emailAddress">0123456789</saml:NameID>
```

```
            <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
              <saml:SubjectConfirmationData NotOnOrAfter="2016-03-28T10:11:53Z"/>
            </saml:SubjectConfirmation>
          </saml:Subject>
          <saml:Conditions NotBefore="2016-03-28T10:09:53Z"
 NotOnOrAfter="2016-03-29T10:11:53Z">
            <saml:AudienceRestriction>
              <saml:Audience>jwt_saml</saml:Audience>
            </saml:AudienceRestriction>
          </saml:Conditions>
          <saml:AuthnStatement AuthnInstant="2016-03-28T10:10:53Z">
            <saml:AuthnContext>
              <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</
saml:AuthnContextClassRef>
            </saml:AuthnContext>
          </saml:AuthnStatement>
          <saml:AttributeStatement>
            <saml:Attribute Name="given_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
              <saml:AttributeValue xsi:type="xs:string">john</saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="email" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
              <saml:AttributeValue xsi:type="xs:string">john.doe@example.com</
saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="iss" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
              <saml:AttributeValue xsi:type="xs:string">isam</saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="family_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
              <saml:AttributeValue xsi:type="xs:string">doe</saml:AttributeValue>
            </saml:Attribute>
          </saml:AttributeStatement>
        </saml:Assertion>
      </wst:Base>
    </ns1:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bold element in the SAML2 assertion is mapped to the JSON attributes in the result.

b) Save this file as `saml2.xml`.

c) Run the following **`curl`** command, where `saml2.xml` is the WS-Trust 1.2 message:

```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary
@saml2.xml https://ip-rte/TrustServer/SecurityTokenService
```

The following results are returned:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-
ENV:Header>
    <soap:Body>
        <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/
ws-sx/ws-trust/200512">
            <wst:RequestSecurityTokenResponse
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
            wsu:Id="uuidc1676e30-0153-16a8-86b5-c34fd1aca7a8">
                <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
                xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                    <wsa:EndpointReference>
                         <wsa:Address>SAML2_AppliesTo</wsa:Address>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <wst:RequestedSecurityToken>

  <JWT>eyJzdWIiOiIwMTIzNDU2Nzg5IiwiZ2l2ZW5fbmFtZSI6ImpvaG4iLCJOb3RPbk9yQWZ0ZXIiOiIyMDE2LTAz

 LTI5VDEwOjExOjUzWiIsIkF1dGhlbnRpY2F0aW9uTWV0aG9kIjoidXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6MS4w

 OmFtOnBhc3N3b3JkIiwiZW1haWwiOiJqb2huLmRvZUBleGFtcGxlLmNvbSIsIkF1ZGllbmNlUmVzdHJpY3Rpb25

 Db25kaXRpb24uQXVkaWVuY2UiOiJqd3Rfc2FtbCIsImlzcyI6ImlzYW0iLCJJc3N1ZUluc3RhbnQiOiIyMDE2LT

 AzLTI4VDEwOjEwOjUzWiIsImZhbWlseV9uYW1lIjoiZG9lIiwiTm90QmVmb3JlIjoiMjAxNi0wMy0yOFQxMDowO
 To1M1oiLCJBdXRoZW50aWNhdGlvbkluc3RhbnQiOiIyMDE2LTAzLTI4VDEwOjEwOjUzWiIsImlzc3VlciI6Iml
                zYW0ifQ==</JWT>
                </wst:RequestedSecurityToken>
                <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</
wst:RequestType>
                <wst:Status>
                    <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/
valid</wst:Code>
                </wst:Status>
            </wst:RequestSecurityTokenResponse>
        </wst:RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

The bold embedded element, **`<JWT> </JWT>)`** , is the result in a Base64 encoded JSON Web Token:

```
{
  "sub": "0123456789",
  "given_name": "john",
  "NotOnOrAfter": "2016-03-29T10:11:53Z",
  "AuthenticationMethod": "urn:oasis:names:tc:SAML:1.0:am:password",
  "email": "john.doe@example.com",
  "AudienceRestrictionCondition.Audience": "jwt_saml",
  "iss": "isam",
  "IssueInstant": "2016-03-28T10:10:53Z",
  "family_name": "doe",
  "NotBefore": "2016-03-28T10:09:53Z",
  "AuthenticationInstant": "2016-03-28T10:10:53Z",
  "issuer": "isam“
}
```

Related tasks

“Mapping a base64 encoded JSON string to a SAML2 token example” on page 168
You can map a base64 encoded JSON string to a SAML 2 token by using a JavaScript mapping rule.

Chapter 7. Nested single sign-on flows

You can nest SAML or OpenID Connect single sign-on flows. That is, you can resume an initial SSO flow after the completion of a second SSO flow.

For SAML, a *nested SSO flow* means the involvement of an IdP proxy between a real service provider (SP) and a real identity provider (IdP). The IdP proxy delegates the user credentials authentication to the real IdP.

For OpenID Connect, a nested SSO flow means the involvement of an OAuth provider (OP) proxy between a real relying party (RP) and a real OP. The OP proxy delegates the user credentials authentication to the real OP.

A nested SSO flow involves the following two SP or RP-initiated SSO flows:

- Between the real SP or RP and an IdP or OP proxy.
- Between the IdP or OP proxy (acts as an SP or RP) and the real IdP or OP.

After the second SSO flow completes the authentication of credentials with the real IdP or OP, the IdP or OP proxy has an implicit mechanism to resume the first SSO flow to sign in to the real SP or RP.

When you install an appliance to work as an IdP proxy, create an identity provider and service provider federation and map them to a single reverse proxy instance. See Chapter 8, “Configuring a reverse proxy point of contact server,” on page 179.

Note: Configure the proper mapping rules in the IdP proxy federations to avoid duplicate attributes in STSUU to attain the successful flow of nested SSO. See the following topics: .

- “Mapping a local identity to a SAML 2.0 token” on page 19
- “Mapping a SAML 2.0 token to a local identity” on page 21
- OpenID Connect Provider mapping rules
- OpenID Connect Relying Party mapping rules

Table 99. Supported nested SSO combinations

First SSO flow	**Second SSO flow**
SAML (HTTPRedirect, HTTPPost, HTTPArtifact)	SAML (HTTPRedirect, HTTPPost, HTTPArtifact)
SAML (HTTPRedirect, HTTPPost, HTTPArtifact)	OpenID Connect
OpenID Connect	OpenID Connect
OpenID Connect	SAML (HTTPRedirect, HTTPPost, HTTPArtifact)

The following supported nested flows are for the authentication delegated to the external IdP during an OAuth20 flow:

Table 100. Supported nested OAuth flow combinations

First flow	**Second flow**
OAuth20	SAML (HTTPRedirect, HTTPPost, HTTPArtifact)
OAuth20	OpenID Connect

Chapter 8. Configuring a reverse proxy point of contact server

Configuring a SAML 2.0 or OpenID Connect federation requires that you set up a reverse proxy instance as the point of contact.

Before you begin

You can use these instructions to configure a reverse proxy instance, or you can use the Web services REST APIs. The REST API topic is located in **Web** > **Manage** > **Reverse Proxy** > **Federation Configuration**.

Note: If you use the Web services REST APIs to configure a reverse proxy instance, ensure that the junction name is `/isam`.

About this task

The reverse proxy instance that you use authenticates users at the identity provider and protects services at the service provider. You must have a reverse proxy instance for both the service provider and the identity provider.

See Reverse proxy instance management for more information.

Procedure

1. Import the federation runtime SSL certificate into the reverse proxy trusted signer certificates keystore. Use the local management interface to import the certificate. See Managing SSL certificates.
2. Using the **pdamin** command, create the `/isam` junction to the federated runtime. Substitute the values of your runtime in the following command:

```
server task hostname-webseal-instanceName create -t ssl -c all -s -b ignore -j
  -e utf8_uri -J inhead -r -q /sps/cgi-bin/query_contents -f
  -h  runtimeHostname -p runtimePort /isam
```

3. Update the reverse proxy configuration file by using the local management interface:
   a) Click **Web** > **Manage** > **Reverse Proxy**.
   b) Select the reverse proxy instance to update, and click **Manage** > **Configuration** > **Edit Configuration File**.
   c) Edit the configuration file with the following stanzas and entries, depending on the federation protocol:

   SAML 2.0

```
[ba]:
ba-auth = none
[forms]:
forms-auth = https
[authentication-levels]:
level = ext-auth-interface
[eai]:
eai-auth = https
retain-eai-session = yes
eai-verify-user-identity = no
eai-redir-url-priority = yes
[eai-trigger-urls]:
trigger = /isam/sps/auth*
trigger = /isam/sps/federation_name/saml20/soap*
trigger = /isam/sps/federation_name/saml20/slo*
trigger = /isam/sps/federation_name/saml20/login*
[session]:
```

```
user-session-ids = yes
```

Legacy OpenID Connect

```
[ba]:
ba-auth = none
[forms]:
forms-auth = https
[junction:/isam]:
reset-cookies-list = *JSESSIONID*,*WAS*
(RP ONLY) [authentication-levels]:
level = ext-auth-interface
(RP ONLY) [eai]:
eai-auth = https
eai-redir-url-priority = yes
(RP ONLY) [eai-trigger-urls]:
trigger = /isam/sps/oidc/client/federation_providerID*
```

OpenID Connect Relying Party

```

[ba]:
ba-auth = none
[forms]:
forms-auth = https
[junction:/isam]:
reset-cookies-list = *JSESSIONID*,*WAS*
[authentication-levels]:
level = ext-auth-interface
[eai]:
eai-auth = https
eai-redir-url-priority = yes
[eai-trigger-urls]:
trigger = /isam/sps/oidc/rp/fedname/redirect/*
```

4. Using the **pdadmin** command, define the nobody, anyauth, and unauth ACLs. Note that the WebSEAL user should be used for default-webseald/isam-op.

```
acl create fedname-nobody
acl modify fedname-nobody set user default-webseald/hostname TcmdbsvaBRl
acl modify fedname-nobody set user sec_master TcmdbsvaBRrxl
acl modify fedname-nobody set group iv-admin TcmdbsvaBRrxl
acl modify fedname-nobody set group webseal-servers Tgmdbsrxl
acl modify fedname-nobody set any-other T
acl modify fedname-nobody set unauthenticated T

acl create fedname-anyauth
acl modify fedname-anyauth set user default-webseald/hostname TcmdbsvaBRl
acl modify fedname-anyauth set user sec_master TcmdbsvaBRrxl
acl modify fedname-anyauth set group iv-admin TcmdbsvaBRrxl
acl modify fedname-anyauth set group webseal-servers Tgmdbsrxl
acl modify fedname-anyauth set any-other Tr
acl modify fedname-anyauth set unauthenticated T

acl create fedname-unauth
acl modify fedname-unauth set user default-webseald/hostname TcmdbsvaBRl
acl modify fedname-unauth set user sec_master TcmdbsvaBRrxl
acl modify fedname-unauth set group iv-admin TcmdbsvaBRrxl
acl modify fedname-unauth set group webseal-servers Tgmdbsrxl
acl modify fedname-unauth set any-other Tr
acl modify fedname-unauth set unauthenticated Tr
```

5. Using the **pdadmin** command, create the ACLs on the policy server, and attach them to the relevant endpoints.

SAML 2.0

```
fedname-nobody:
/WebSEAL/hostname-webseal/isam
fedname-unauth:
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/login
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/sloinitial
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/mnids
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/logininitial
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/slo
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/soap
fedname-anyauth:
```

```
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/mnidsinitial
/WebSEAL/hostname-webseal/isam/sps/fedname/saml20/auth
/WebSEAL/hostname-webseal/isam/sps/wssoi
/WebSEAL/hostname-webseal/isam/sps/auth
```

Legacy OpenID Connect

```
fedname-nobody:
/WebSEAL/hostname-instance/isam
fedname-unauth:
/WebSEAL/hostname-instance/isam/sps/static
/WebSEAL/hostname-instance/isam/sps/fedname/oidc/auth
/WebSEAL/hostname-instance/isam/oidc/scripts
/WebSEAL/hostname-instance/isam/oidc/endpoint/amapp-runtime-fedname/token
/WebSEAL/hostname-instance/isam/oidc/endpoint/amapp-runtime-fedname/introspect
/WebSEAL/hostname-instance/isam/oidc/endpoint/amapp-runtime-fedname/authorize
(RP Only) /WebSEAL/hostname-instance/isam/sps/oidc/client/fedname
(RP Only) /WebSEAL/hostname-instance/isam/oidcclient/redirect
fedname-anyauth
/WebSEAL/hostname-instance/isam/sps/auth
```

OpenID Connect Relying Party

```
fedname-unauth:
/WebSEAL/hostname-instance/isam/sps/oidc/rp/fedname/kickoff
/WebSEAL/hostname-instance/isam/sps/oidc/rp/fedname/redirect
```

6. Using the **pdamin** command, add the `HTTP-Tag-Value` attribute to the `/isam` junction object to propagate the `user_session_id` to the federation runtime:

   - If `force-tag-value-prefix = yes`:

     ```
     object modify /WebSEAL/hostname-default/isam set attribute
       HTTP-Tag-Value user_session_id=USER-SESSION-ID
     ```

   - If `force-tag-value-prefix = no`:

     ```
     object modify /WebSEAL/hostname-default/isam set attribute
       HTTP-Tag-Value tagvalue_user_session_id=USER-SESSION-ID
     ```

Chapter 9. Global settings

You can use the LMI to access an administrative menu to configure global settings that are used by both Federation and Advanced Access Control.

The Local Management Interface (LMI) has a user interface page for administering each major feature in IBM Security Verify Access. Since some features are used by multiple licensing levels for the product, the administration page for these features can be accessed through multiple user interface menu paths.

You can use either of the following LMI menus to access the global settings:

- **AAC** > **Global Settings**
- **Federation** > **Global Settings**

You can use the global settings menus to configure the following features:

- Advanced Configuration

  Some of the advanced configuration properties are common to Advanced Access Control and Federation. Others are specific to one of the licensing levels.
- User Registry

  Use these settings to administer users and group memberships for the user registry that is used by the runtime applications. Management tasks are common to Advanced Access Control and Federation.
- Runtime Parameters

  You can use the Runtime Parameters menu to view runtime status, tune runtime parameters, and set tracing on the runtime. These functions are common to Advanced Access Control and to Federation.

  In addition, the runtime tracing feature can be set in the LMI through **Monitor > Logs > Runtime Tracing** > ..

  The topic for Runtime Parameters is also included in the appliance troubleshooting section of the IBM Knowledge Center. See Tuning runtime application parameters and tracing specifications
- Template Files

  Template files are HTML pages that are presented to your users. You can customize the content of the pages for your deployment by setting supported macros, or by adding JavaScript scripting. Template pages are used in multiple scenarios.

  - Customizing the authentication process, such as error messages
  - Specifying settings for the supported authentication mechanisms
  - Customizing error messages for authentication attempts
  - Obtaining consent for registering devices
  - Specifying authorization parameters for OAuth 2.0
  - Configuring user self-care tasks
- Mapping Rules

  Mapping rules are JavaScript code that runs during the authentication flow for Advanced Access Control and Federation. Mapping rules can be used for multiple purposes. For Advanced Access Control, you can modify rules for the Authentication Service, OTP, and OAuth 2.0. For Federation, you can modify mapping rules to manage identities for OIDC and SAML 2.0.

- Distributed Session Cache

  The Distributed Session Cache is supplied by the Web Reverse Proxy and is used with all activation levels. The management windows in the LMI can also be accessed through **Web** > **Manage** > **Distributed Session Cache**.

  For an overview of the Distributed Session Cache, and a review of advanced configuration options, see: Distributed Session Cache.
- Server Connections

  Advanced Access Control and Federations both use the IBM Security Verify Access appliance to connect to external data sources. For Advanced Access Control, you can use the server connections menus to configure LDAP or database server connections so that you can set up policy information points. For Federation, you can configure an LDAP server as an attribute source for attribute mapping.
- Point of Contact

  IBM Security Verify Access provides servers, such as WebSEAL, that function as point of contact servers for handling external requests for authentication and authorization. You can configure a point of contact profile to specify the information that is needed for the runtime to communicate with a specific point of contact server. Security Verify Access provides three Point of Contact profiles that are ready for use. You can specify callback parameters and values for these profiles.
- Access Policies

  You can use access policies to perform step-up and re-authentication during a single sign-on flow based on contextual information. Access policies can be enforced at a federation or at API Protection for OAuth and OpenID Connect.

Note: The LMI mega-menu for the **Web** licensing level also presents a set of tasks under a **Global Settings** heading. These tasks are different from the tasks under the **Global Settings** menu for **AAC** and **Federation**. The **Web > Global Settings** LMI menus are not used with **AAC** and **Federation** .

Managing advanced configuration

Adjust configuration settings in supported configuration files.

About this task

The advanced configuration includes both properties and files. The properties configuration panel displays a table of configuration settings. Some can be modified and some are read-only. Each setting is displayed as a row in the table. The name of the setting is listed in the *key* column. The current value of the key is listed in the *value* column. You can locate a setting by using one of the following methods:

- Scroll through the list until you see the setting.

  By default, all configuration settings are included in the list.
- Filter the list by entering a string in the **Filter** field.

  When you enter a string, the list is modified to show only the settings that contain the specified string.
- Filter the list by selecting a category from the **Filter by Category** menu.

For descriptions of the categories and properties, see "Advanced configuration properties" on page 185.

This files configuration panel displays a table of configuration files. Each file can be viewed, edited, replaced, or exported.

Procedure

1. Select the menu entry for your licensing level:
   - If using an Advanced Access Control license, select **AAC** > **Global Settings** > **Advanced Configuration**.
   - If using a Federation license, select **Federation** > **Global Settings** > **Advanced Configuration**

2. To edit a property key, select the edit icon for the key.

   Note: You cannot edit keys that are marked with the read-only icon: .

   When you choose to edit a key, a new window displays the name of the key and the current value.

   a) Edit the value of your deployment.
   b) Click **OK**.
3. To manage an advanced configuration file click the Files link in the page header and click the required file in the table.
   a) To edit the file click the **Edit** button to make the required changes. Click **Save**.
   b) To replace the existing file click the **Import** button in the Manage drop down, select the new file in the dialog and click Import.
   c) To export the existing file click the **Export** button in the Manage drop down.
4. Click **OK**.
5. Deploy the changes.

Advanced configuration properties

Modify the advanced configurations for Advanced Access Control or Federation to meet the requirements of your organization.

Category filter

The category filter displays names of grouping of configuration settings. The groupings correspond to functional areas. When you select a category, the user interface displays only the settings for the category.

Table 101. Filter by Category

Category	**Displays values for:**
All	All keys
poc.websealAuth	“WebSEAL Authenticate Callback” on page 186
poc.otpAuth	“One-time password Authenticate Callback” on page 187
poc.authPolicy	“Authentication-Policy Callback” on page 187
sps.httpRequestClaims	“SPS HTTP request claims” on page 187
distributedMap	“Distributed shared data storage” on page 187
userBehavior	“Attribute matcher properties” on page 188
ipReputation	“IP reputation PIP properties” on page 188
attributeCollection	“Attribute collector properties” on page 188
deviceRegistration	“Device registration properties” on page 190
runtime	“Runtime properties” on page 191
sps.page	“SPS page” on page 192

Table 101. Filter by Category (continued)

Category	**Displays values for:**
sps	"Single sign-on protocol service" on page 191
riskEngine	"Risk engine properties" on page 193
sps.authService	"Authentication service properties" on page 193
authsvc.stateMgmt	"Authentication service session store properties" on page 194
session	"Session" on page 195
distributedSessionCache	"Distributed session cache" on page 195
otp.retry	"TOTP and HOTP retry properties" on page 196
oauth20	"OAuth20" on page 197
util.httpClient	"HTTP client" on page 198
util.httpClient v2	"HTTP Client version 2" on page 199
demo	"Demo" on page 204
knowledge.questions	"Knowledge questions properties" on page 204
kess	"Key encryption and signing service (KESS)" on page 204
jwks	"JSON Web Key" on page 206
pip	"Policy information point (PIP)" on page 206
sts	"Security token service (STS)" on page 206
mmfa	Mobile Multi-Factor Authentication (MMFA)
wsfed	"WS-Federation" on page 208
saml20	"SAML 2.0" on page 209
demo	"Demo" on page 204
saml11	"SAML 1.1" on page 208
oidc	"OIDC" on page 210
js	"Rhino Javascript Engine" on page 210

WebSEAL Authenticate Callback

`poc.websealAuth.authLevel`

The authentication level of the callback.

Data type: Integer

Example: 1

One-time password Authenticate Callback

`poc.otp.authLevel`

The authentication level of the callback.

Data type: Integer

Example: 2

`poc.otp.backwardCompatibilityEnabled`

Indicates whether the one-time password authentication mechanism should run in backward compatibility mode. The default value is `false` if it is a new installation. The default value is `true` if the installation is an upgrade.

Data type: Boolean

Example: `true`

Authentication-Policy Callback

`poc.authPolicy.allowRequestOverride`

Whether the authentication level, the authentication mode, and the authentication type of the callback can be overwritten by query string parameters.

Data type: Boolean

Example: `true`

`poc.authPolicy.authLevel`

The authentication level of the callback.

Data type: Integer

Example: 1

`poc.authPolicy.authType`

The authentication type of the callback.

Data type: String

Example: `COMPLEMENTARY`, `HIERARCHICAL`

SPS HTTP request claims

`sps.httpRequestClaims.enabled`

Whether HTTP request information is sent to STS as `HTTPRequestClaims`. This flag additionally makes HTTP Request attributes (Headers, Cookies and Parameters) available to administrators in OIDC, OAuth, and SAML (see HTTP Claims in OIDC, OAuth and SAML JavaScript Mapping Rules), Authsvc and InfoMap (see HTTP Claims in Authsvc and InfoMap JavaScript Mapping Rules) and FIDO2 (see HTTP Claims in FIDO2 Mediator JavaScript Mapping Rules) JavaScript Mapping rules.

Data type: Boolean

Example:`false`

`sps.httpRequestClaims.filterSpec`

The filter that specifies the HTTP request information that is sent to STS as `HTTPRequestClaims`.

Data type: String

Example: `cookies=*:headers=*`

Distributed shared data storage

`distributedMap.cleanupWait`

The amount of time, in milliseconds, to wait before it performs another cleanup against the distributed map.

Distributed map clean up can be disabled by setting the `cleanupWait` to 0.

Data type: Integer

Example: `10000`

`distributedMap.defaultTTL`
The amount of time, in seconds, that the entries in the distributed map must live when no lifetime is specified for an entry.

Data type: Integer

Example: `3600`

`distributedMap.getRetryDelay`
The amount of time, in milliseconds, to wait before it performs another retrieval against the distributed map. The default is `0`.

Data type: Integer

Example: `500`

`distributedMap.getRetryLimit`
The number of retrievals that is done against the distributed map before it returns that the retrieved data is not in the distributed map. The default is `0`.

Data type: Integer

Example: `10`

Attribute matcher properties

`userBehavior.minimumUsageHistoryRequired`
Minimum usage data records required for any usage data analysis; used by LoginTimeMatcher.

Data type: Integer

Example: `8`

`userBehavior.ipAddressRequestAttribute`
The XACML request attribute to read from the IP address.

Data type: String

Example: `urn:ibm:security:subject:ipAddress`

IP reputation PIP properties

`ip.reputation.ipAddressAdverseReputationThreshold`
The value that an IP classification score must be at or above for an IP address to be considered as that classification.

Data type: Integer

Example:`50`

`ipReputation.dbConnectionTimeout`
Indicates the number of seconds that the IP reputation policy information point (PIP) waits for a connection to the IP reputation database. The `ipReputation.dbConnectionTimeout` property defaults to `120`.

Data type: Integer

Example: `60`

Attribute collector properties

`attributeCollection.cookieName`
Correlation ID used by the attribute collector.

Data type: String

Example: `ac.uuid`

`attributeCollection.requestServer`
Request server for attribute collector. A list of the allowable hosts where the ajaxRequest can be sent from.

Data type: String List

Example: `https://rbademo.example.com,https://rbaemo2.example.com`

`attributeCollection.serviceLocation`
Location of the attribute collector.

Data type: String List

Example: `http://rbademo.example.com/mga`

`attributeCollection.sessionTimeout`
Number of seconds in which sessions stored in context-based access will automatically expire, unless updated. If any attribute in the session is updated, the session expiry is extended by the specified number of seconds configured in this property. The default is `1800` seconds.

Data type: Integer

Example: `1800 seconds`

`attributeCollection.enableGetAttributes`
Enables the REST GET method to return attributes.

Data type: Boolean

Example: `false`

`attributeCollection.getAttributesAllowedClients`
A comma-separated list of clients that are allowed to access the ACS REST GET method.

If this property is not set and `attributeCollection.enableGetAttributes` is set to `true`, anyone can access the GET method. If this property is set but `attributeCollection.enableGetAttributes` is set to `false`, this property is ignored.

Data type: String List

Example: `hostname1, hostname2`

`attributeCollection.hashAlgorithm`
The algorithm that is used to create the hash.

Data type: String

Example: SHA256

`attributeCollection.attributesHashEnabled`
A comma-separated list of attribute URI values configured for hashing.

Attention: Do not hash the following attributes:

- `ipAddress`
- `geoLocation`
- `accessTime`

Data type: String List

Example:

```
urn:ibm:security:environment:http:userAgent,
urn:ibm:security:environment:deviceFonts,
urn:ibm:security:environment:browserPlugins
```

`attributeCollection.authenticationContextAttributes`
Comma-separated lists of attribute names to be collected during an authentication service obligation. The maximum number of characters for this property is 200.

Data type: String List

Example: `authenticationLevel, http:host`

Device registration properties

`deviceRegistration.allowIncompleteFingerprints`

Specifies to allow the device registration obligation to store fingerprints where all the fingerprint attributes are not available on the session information.

Data type: Boolean

Example: `false`

`deviceRegistration.checkForExpiredDevices`

Determines whether registered devices are inactive or expired. If the `deviceRegistration.checkForExpiredDevices` property is set to `true`, the risk engine checks whether a device is inactive or expired. The `deviceRegistration.checkForExpiredDevices` property defaults to `false`, which means that users can use any of the devices that are registered.

Date type: Boolean

Example: `true`

`deviceRegistration.cleanupThread.batchSize`

Specifies if batch delete is enabled for expired devices and how many records are deleted per batch.

If the value is defined as `0` or is blank, batch delete is not enabled and all expired devices are deleted using one SLQ delete statement.

If the value is defined as an integer greater than `0`, batch delete is enabled. The number that you specify determines how many records are deleted in each batch. The batch delete continues until all of the expired devices are deleted. The batch process is useful for deleting a large quantity of expired devices.

Data type: Integer

Example: `1000` (Batch delete is enabled, with a batch size of 1000 records.)

`deviceRegistration.deviceMatchThreshold`

The risk score threshold where an existing fingerprint is considered to match the incoming device fingerprint.

Data type: Integer

Example: `20`

`deviceRegistration.inactiveExpirationTime`

Specifies the number of days that a device must be inactive for it to expire. The `deviceRegistration.inactiveExpirationTime` property defaults to `90`.

Date type: Integer

Example: `100`

`deviceRegistration.maxRegisteredDevices`

Maximum device fingerprint count. The default is `10`. Valid values are `1` to `100`.

Data type: Integer

Example: `10`

`deviceRegistration.maxUsageDataPerUser`

Maximum number of historical usage attribute records stored per user. The default is `200`. Valid values are `1` to `5000`.

Data type: Integer

Example: `1000`

`deviceRegistration.permitOnIncompleteFingerprints`

Specifies to permit access to the resource if the fingerprint collected by the device registration obligation does not include all fingerprint attributes.

Data type: Boolean

Example: `false`

Runtime properties

`runtime.dbLoggingEnabled`

Enables fine-grained logging for database SQL statements.

Data type: Boolean

Example: `false`

`runtime.hashAlgorithm`

The algorithm that is used for hashing. The supported algorithms are:

- SHA-1
- SHA-256
- SHA-384
- SHA-512

The `runtime.hashAlgorithm` property defaults to `SHA-256`.

Data type: String

Example: `SHA-256`

`runtime.verificationHashAlgorithms`

Defines the hashing algorithms that are used to verify a hashed value. The value is typically a comma separated list of hashing algorithms.

Data type: String

Example: `SHA-256, SHA-1`

Single sign-on protocol service

`sps.setCookiesAsSecure`

Determine whether to flag the cookies set by Security Verify Access as secure.

The default value is `false`.

Data type: Boolean

Example: `false`

`sps.targetURLWhitelist`

Specifies a list of allowed target URLs for SAML 2.0, OpenID Connect, and the authentication service. Use this property to prevent an attacker from redirecting a user to malicious target URLs.

The value of this advanced configuration property is a comma-separated string, where each string is a target URL in the form of a regular expression. The regular expression must not contain commas, and spaces between regular expressions are ignored.

- For SAML 2.0 SSO flows, you can specify a Target URL when you configure the initial URL in flows that are initiated by either the Identity Provider or the Service Provider. For more information, see "SAML 2.0 profile initial URLs" on page 14.
- For Open ID Connect flows, you can specify a Target URL when you configure the initial URL for Relying Party initiated single sign-on. For more information, see Relying Party SSO initiation endpoint.
- For the authentication service, you can specify a Target URL when you configure the authentication service trigger URL. For more information, see Configuring authentication.

The default value is ".*".

Data type `String`

Example

```
(http|https)://www.app.ibm.com/.*, (http|https)://www.myidp.ibm.com/.*
```

`sps.illegalUrlSubstrings`

A comma-separated list of strings, the single sign-on service stops processing the request if the request URL query parameters contain any of the strings.

The default value is "".

Data type: String

Example:

```
"<script"
```

`sps.doNotSendXFrameOptionsHeader`

Specifies whether an X-Frame-Options header with value SAMEORIGIN must be returned from the SPS endpoints for browser based flows. When this property is set to `true`, no X-Frame-Options header is sent.

Note: The `sps.doNotSendXFrameOptionsHeader` property defaults to false.

Data type: Boolean

Example: False

SPS page

`sps.page.htmlEscapedMacros`

A comma-separated list of macros that is HTML-escaped when it is rendered in pages that are sent to the browser.

Data type: String

Example:

```
@REQ_ADDR@,
@DETAIL@,
@EXCEPTION_STACK@,
@EXCEPTION_MSG@,
@OTP_METHOD_ID@,
@OTP_METHOD_LABEL@,
@OTP_HINT@,
@ERROR_MESSAGE@,
@MAPPING_RULE_DATA@
```

`sps.page.exceptionMacros`

A comma-separated list of `classname:macro` pairs. `Classname` is the fully qualified name of the exception class. Macro is the name of the macro to which the class maps.

Data type: String

Example:

```
com.tivoli.am.fim.otp.deliveries.OTPDeliveryException:@OTP_DELIVERY_EXCEPTION@,
com.tivoli.am.fim.otp.providers.OTPProviderException:@OTP_PROVIDER_EXCEPTION@
```

`sps.page.notEscapedMacros`

A comma-separated list of macros that are **not** HTML-escaped when they are rendered in pages that are sent to the browser. Macros that do not appear in this list or the Macros in the **`htmlEscapedMacros`** list are HTML-escaped.

Data type: String

Example:

```

@COOKIE_NAME@,
@SERVER_NAME@,
@JUNCTION@
```

`sps.page.hiddenMacros`

A comma-separated list of macros that are not rendered in the pages that are sent to the browser. The default value is `@EXCEPTION_STACK@`.

Data type: String

Example: `@EXCEPTION_STACK@`

Risk engine properties

`riskEngine.reportsEnabled`

Enables the generation of risk calculation reports.

Data type: Boolean

Example: `false`

`riskEngine.reportsMaxStored`

Specifies the maximum number of reports to store.

Data type: Integer

Example: 5

Authentication service properties

`sps.authService.reauthenticationEnabled`

Specifies that the authentication service performs authentication even if the user already has an authenticated session at the required authentication level.

Data type: Boolean

Example: `true`

`sps.authService.policyKickoffMethod`

Specifies whether the URLs `/sps/authsvc` and `/sps/apiauthsvc` can be invoked with the **`policyId`** query string parameter.

If set to `query`, the authentication service endpoints continue to accept **`policyId`** as a query or post parameter.

If set to `path`, authentication service endpoints are changed to:

- `/sps/apiauthsvc/policy/<shortPolicyId>`
- `/sps/authsvc/policy/<shortPolicyId>`

Where `<shortPolicyId>` is the value that comes after the prefix `urn:ibm:security:authentication:asf:`

By default, the value is set to `both`.

When set to `both`, either the `path` or `query` parameter can be used to initiate an authentication service flow.

`sps.authService.stateIdSource.authsvc`

Specifies whether the URL `/sps/authsvc` can be invoked with the **`StateId`** query string parameter.

If set to **Body and Query**, the authentication service endpoint continues to accept StateId as a query or body parameter.

If set to **Body Only**, the authentication service endpoint only accepts the StateId as a body parameter (POST or PUT).

Data type: String

Default: Body and Query

Example: Body only

`sps.authService.stateIdSource.apiauthsvc`

Specifies whether the URL `/sps/apiauthsvc` can be invoked with the **`StateId`** query string parameter.

If set to **Body and Query**, the API authentication service endpoint continues to accept StateId as a query or body parameter.

If set to **Body Only**, the API authentication service endpoint only accepts the StateId as a body parameter (POST or PUT).

Data type: String

Default: Body and Query

Example: Body Only

Authentication service session store properties

`authsvc.stateMgmt.cookieless`

Enables the server side storage of session data for the authentication service. If enabled, this removes the need for the JSESSIONID cookie.

Data type: Boolean

Example: `true`

Default value: `true`

`authsvc.stateMgmt.store`

Specifies the storage type that is used by the Authentication service to cache user session data. The authentication service can be supported by the DSC (Session), the HVDB, or stored in Memory.

Note: For clustered environments, storage in Memory does not replicate between nodes.

Data type: String

Example: `Memory`

Default value: HVDB

`authsvc.stateMgmt.HVDB.lifetime`

Length of time in seconds that a session is cached for. Once this time period is exceeded, the user's session is removed from the session store. If this value is less than 0, the default lifetime of 3600 seconds (1 hour) is enforced. This configuration option applies only to session stores supported by the HVDB or Memory.

Data type: Integer

Example: 60 (1 minute)

Default value: `3600`

`authsvc.stateMgmt.HVDB.maxSessions`

Maximum number of user sessions to be cached at any point in time. If the number of sessions in the store exceeds this value, the oldest session is invalidated. This configuration option only applies to session stores that are supported by the HVDB or Memory.

Data type: Integer

Example: `10000`

Default value: `1000`

`authsvc.stateMgmt.HVDB.cleanupWait`

Frequency (in seconds) that expired or excess sessions are removed from the session store. Setting this entry to -1 disables the cleanup thread. This configuration option only applies to session stores backed by the HVDB or Memory.

Data type: Integer

Example: `30`

Default value: `120`

`authsvc.stateMgmt.HVDB.cleanupThread.batchSize`
Maximum number of expired sessions which are removed in a single cleanup operation. If the value is defined as 0 or is blank, batch delete is not enabled. All expired sessions are deleted by using one SLQ delete statement. If the value is defined as an integer greater than 0, batch delete is enabled. The number that you specify determines how many sessions are deleted in each batch. The batch delete continues until all of the expired sessions are deleted. This configuration option only applies to sessions that are stored in the HVDB or Memory.

Data type: Integer

Example: `1000`

Default value: `0`

`authsvc.stateMgmt.HVDB.cleanupOnlyOnPrimaryMaster`
Prevent the cleanup thread from running on non-primary master nodes in a clustered environment. This configuration option only applies to sessions that are stored in the HVDB or Memory.

Data type: Boolean

Example: `true`

Default value: `true`

Session

`distributedSessionCache.enabled`
A switch that dictates if the distributed session cache is used for session failover. If this setting is not enabled, the distributed session cache server still runs as a service, but the client does not use it.

Data type: Boolean

Example: `false`

`distributedSessionCache.localCacheSize`
The number of sessions to be stored on the client as a local cache. A value of 0 or less means that any number of sessions can be cached by the client. A low number requires more connections to the distributed session cache if there are many active sessions. A high number runs the risk of running out of memory if many sessions are locally cached. All sessions are still stored on the distributed session cache when it is enabled.

Data type: Integer

Example: `4096`

`session.dbCleanupInterval`
Specifies the interval, in seconds, that the database cleanup thread runs to remove expired data in the runtime database. The default is `86400`. The minimum value for this property is `3600`. For more information, see Runtime database tuning parameters

Session database clean up can be disabled by setting the `dbCleanupInterval` to `0`. This is not overridden by the minimum value.

Data type: Integer

Example: `90000`

Distributed session cache

`distributedSessionCache.enabled`
A switch that dictates if the distributed session cache is used for session failover. If this setting is not enabled, the distributed session cache server still runs as a service, but the client does not use it.

Data type: Boolean

Example: `false`

`distributedSessionCache.localCacheSize`
The number of sessions to be stored on the client as a local cache. A value of 0 or less means that any number of sessions can be cached by the client. A low number requires more connections to the distributed session cache if there are many active sessions. A high number runs the risk of running out of memory if many sessions are locally cached. All sessions are still stored on the distributed session cache when it is enabled.

Data type: Integer

Example: 4096

`distributedSessionCache.externalServers`

A list of locations of the distributed session cache servers in weighted order.

Syntax:

```
<primary_address>:<port>[:<ssl>];<secondary_address>:<port>[:<ssl>],...
```

`<address>`

The IP address of the distributed session cache server. For example, 10.150.21.80.

`<port>`

The port for the distributed session cache. For example, 2126.

`<ssl>`

Whether SSL communication with the distributed session cache is required. The default value is false.

Data type: String

Example:

```
10.150.21.80:2126:true;10.150.21.81:2126:false,10.150.21.82:2126
```

distributedSessionCache.localCacheEnabled
A switch that dictates whether a local cache of distributed sessions is maintained. If this setting is disabled a higher load is placed on the distributed session cache server. The local cache should only be enabled if all requests from the same client is guaranteed to be sent to the same runtime server (otherwise known as stickiness). Session inconsistencies might occur if the local cache is enabled and stickiness is not maintained. All sessions are still stored in the distributed session cache when it is enabled.

Data type: Boolean

Example: False

TOTP and HOTP retry properties

`otp.retry.enabled`
Whether the retry protection is enabled.

Data type: Boolean

Example: `true`

`otp.retry.maxNumberOfAttempts`
The maximum number of strikes the users can have before they are prevented from logging in.

Data type: Integer

Example: 5

`otp.retry.otpRetryTimeout`
The number in seconds a strike lasts.

Data type: Integer

Example: `600`

OAuth20

`oauth20.clientDataToInclude`

Specifies the OAuth client information to be returned as JSON data. This property is a comma-separated list of the JSON Keys. Valid values are:

```
contact_type
email_address
contact_person
company_name
company_url
phone_number
other_info
```

You can specify one or more of these keys for this property.

Note: The `oauth20.clientDataToInclude` property defaults to `contact_type, email_address, contact_person, company_name, company_url, phone_number, other_info`.

Data type: String

Example: `contact_type, email_address, company_name`

`oauth20.doNotSendXFrameOptionsHeader`

Specifies whether an X-Frame-Options header with value `SAMEORIGIN` must be returned from the OAuth 2.0 endpoints. When set to `true`, no X-Frame-Options header is sent.

Note: The `oauth20.doNotSendXFrameOptionsHeader` property defaults to `false`.

Data type: Boolean

Example: `false`

`oauth20.hashedTokenStorageEnabled`

Enables hashed storage when set to `true`. The Security Verify Access appliance can persist OAuth 2.0 tokens in the clear text form or in the more secure hashed form.

The hashing algorithm set in the `runtime.hashAlgorithm` property will be used. When verifying hashed tokens, the `runtime.verificationHashAlgorithms` property will be used. The algorithms listed in the `runtime.verificationHashAlgorithms` property will be tried in the specified order. This mechanism allows for upgrading of the hashing algorithm while continuing to support old tokens.

Note: The `oauth20.hashedTokenStorageEnabled` property defaults to `false`, and the OAuth 2.0 tokens will be stored as-is.

Data type: Boolean

Example: `false`

`oauth20.sessionEndpointEnabled`

Enables the ability to return an authenticated session at the point-of-contact when the `oauth20.sessionEndpointEnabled` property is set to `true`.

Note: The `oauth20.sessionEndpointEnabled` property defaults to `false`.

Data type: Boolean

Example: `false`

`oauth20.tokenCache.cleanupWait`

The amount of time, in seconds, to wait before it performs another cleanup of expired tokens in the OAuth 2.0 token cache.

Note: The `oauth20.tokenCache.cleanupWait` property defaults to `120`.

OAuth token clean up can be disabled by setting the `cleanupWait` value to 0.

Data type: Integer

Example: `120`

`oauth20.legacyAttributeHandling`

Changes how associated attributes function across the API Protection and OpenID Connect solution. This includes:

- `OauthMappingExtUtils.retrieveAllAssociations()` `OauthMappingExtUtils.getAssociation()` calls in mapping rules
  - When it is set to **True**, it does not return READONLY or SENSITIVE attributes.
  - When it is set to **False**, it returns READONLY or SENSITIVE attributes.
- The user self care endpoint `/mga/sps/mga/user/mgmt/grant/`
  - When it is set to **True**, attributes that are both READONLY and SENSITIVE are returned
  - When it is set to **False**, attributes that are both READONLY and SENSITIVE are not returned.
- Attributes which are saved from attribute sources when performing identity enrichment.
  - When it is set to **True**, attributes are saved against the grant as neither READONLY or SENSITIVE.
  - When it is set to **False**, attributes are saved against the grant as READONLY. The post token rule can be used to update this value if necessary.

HTTP client

`util.httpClient.defaultTrustStore`

Stores the default truststore that HTTPS connections in HTTP client uses.

Note: The `util.httpClient.TrustStore` property defaults to `rt_profile_keys`.

Data type: String

Example: `rt_profile_keys`

`util.httpClient.defaultSSLProtocol`

Stores the default SSL protocol configuration that HTTPS connections in HTTP client uses.

Note: The `util.httpClient.defaultSSLProtocol` property defaults to TLS.

Data type: String

Example: TLS

`util.httpClient.maxActiveConnections`

Specifies the maximum number of HTTP and HTTPS connections, per host, between the appliance runtime and other modules. In a multiple host environment, the runtime might need to establish many HTTP/HTTPS connections at the same time. By specifying this property, you can limit the number of active connections for each host. This setting ensures that each host can obtain their fair share of HTTP/HTTPS connections without being forced to wait for other hosts to release connections.

- Data type: String
- Default: An unlimited number of HTTP/HTTPS connections are permitted

You can specify the maximum number of active connections in one of two ways:

- Specify a maximum number to apply to every host. Syntax:

```
"*=<count>"
```

- Specify a maximum number on a per host basis. Syntax:

  ```
  "<host1>:<port1>=<count>,<host2>:<port2>=<count>,*=<count>"
  ```

  <host>
  The host value can be either an IP address, a hostname or domain name as specified in the Endpoint URL. Specify the host value based on the URL format. For example:

  – IP Address: `192.168.102.192`

  – Hostname or domain name: `www.server1.com`

  <port>=<count>
  The communication port on the host. For example, to limit port 80 to only 100 connections, enter `80=100`.

  `*=<count>`
  The count limit for servers that are not specified by a *`<host>`* value in this property. When set to zero (`*=0`) there is no limit on the number of HTTP/HTTPS connections that can be created to other servers. When set to an integer greater than zero, the integer specifies the maximum number of HTTP/HTTPS connections that can be created to each of the other servers.

  Note: Ensure that `<count>` is specified as a value of type *integer*. Do not use values of type *string* for `<count>`.

Example 1: Specifying a maximum number to apply to every host

For example, your deployment must establish connections to two servers. You want to limit the number of connections to 100 per server. You also want to ensure that when you add additional servers, the number of connections to each additional server is limited to 100.

Use the syntax `"*=<count>"`. For this example:

```
"*=100"
```

Example 2: Specifying maximum numbers on a per host basis

For example, your deployment must establish connections to two servers. You want to limit the number of connections for one server to 100, but allow the other server to have 200 connections. In addition, you do not want to limit the number of connections for any additional servers.

Use the syntax: `"<host1>:<port1>=<count>,<host2>:<port2>=<count>,*=<count>"`

For example, the runtime might need to establish the connections to the following URLs, for an SMS OTP flow and an OIDC flow:

- http://www.server1.com/isam/sms_otp
- https://192.168.102.192/isam/oidc_sts

Example configuration entry:

```
"www.server1.com:80=100,192.168.102.192:443=200,*=0"
```

The example configuration entry specifies:

- The maximum number of HTTP/HTTPS connections that can be created to `www.server1.com` at a time (on port 80) is 100.
- The maximum number of HTTP/HTTPS connections that can be created to `192.168.102.192` at a time (on port 443) is 200.
- There is no limit on the number of HTTP/HTTPS connections that can be created to other hosts.

HTTP Client version 2

`util.httpClientv2.getConnectionTimeout`
Specifies the timeout for retrieving a connection from the connection pool. Value is in seconds.

Note: The `util.httpClientv2.getConnectionTimeout` property defaults to 5 seconds for every host (*=5)

Data type: String

Example: `*=5`

You can specify the timeout by using one of the following methods:

- Specify a timeout that applies to every host and port.

```
"*=<timeout>"
```

- Specify a timeout on a per host and port basis

```
"<host1>:<port1>=<timeout>,<host2>:<port2>=<timeout2>,*=<timeout3>
```

`util.httpClientv2.connectTimeout`

Specifies the timeout for establishing a connection with the remote host. Value is in seconds.

Note: The `util.httpClientv2.connectTimeout` property defaults to 5 seconds for every host (*=5).

Data type: String

Example: `(*=5)`

You can specify the timeout by using one of the following methods:

- Specify a timeout that applies to every host and port.

```
"*=<timeout>"
```

- Specify a timeout on a per host and port basis

```
"<host1>:<port1>=<timeout>,<host2>:<port2>=<timeout2>,*=<timeout3>
```

`util.httpClientv2.connectionInactiveValidate`

Specifies the period of inactivity in milliseconds after which pooled connections must be re-validated prior to being reused. Value is in seconds.

Note: The `util.httpClientv2.connectionInactiveValidate` property defaults to 2 seconds for every host (*=2).

Data type: String

Example: `*=2`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
"*=<value>"
```

- Specify a value on a per host and port basis

```
"<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

`util.httpClientv2.connectionTimeToLive`

Specifies the maximum time a connection stays open. After which it automatically closes. Value is in seconds.

Note: The `util.httpClientv2.connectionTimeToLive` property defaults to no timeout.

Data type: String

Example: `*=30`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

`util.httpClientv2.socketTimeout`

Specifies the timeout to wait for packets to arrive on an established connection. Value is in seconds.

Note: The `util.httpClientv2.socketTimeout` property defaults to 5 seconds for every host (*=5).

Data type: String

Example: `*=5`

You can specify the timeout by using one of the following methods:

- Specify a timeout that applies to every host and port.

```
“*=<timeout>”
```

- Specify a timeout on a per host and port basis

```
“<host1>:<port1>=<timeout>,<host2>:<port2>=<timeout2>,*=<timeout3>
```

`util.httpClientv2.defaultSSLProtocol`

Specifies the default SSL protocol configuration that HTTPS connections in HTTP client uses.

The following values are valid:

- TLSv1
- TLSv1.1
- TLSv1.2
- TLS (This value enables all of the above protocols)

Note: The `util.httpClientv2.defaultSSLProtocol` property defaults to TLS.

Data type: `String`

Example: TLS

`util.httpClientv2.defaultTrustStore`

Specifies the default truststore that HTTPS connections in HTTP client uses.

Note: The `util.httpClientv2.defaultTrustStore` property defaults to `rt_profile_keys`.

Data type: `String`

Example: `rt_profile_keys`

`util.httpClientv2.disableAutoRetries`

Specifies whether or not to disable automatic request recovery and re-execution.

Note: The `util.httpClientv2.disableAutoRetries` property defaults to `false` for every host (*=false).

Data type: String

Example: `*=false`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

util.httpClientv2.enableHostNameVerification

Specifies whether or not to enable hostname verification. If enabled it verifies that the target hostname matches the names that are stored inside the server’s X.509 certificate once the connection is established.

Note: The `util.httpClientv2.enableHostNameVerification` property defaults to `true` for every host (*=host).

Data type: String

Example: `*=true`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

util.httpClientv2.disablePublicSuffixVerification

Specifies whether or not to disable hostname verification using the list of valid public suffixes. HttpClient uses the public suffix list to ensure that wildcards in SSL certificates cannot be misused to apply to multiple domains with a common top-level domain. The HTTP Client ships with a copy of the list retrieved at the time of the release. The local copy is a configuration file named `local-copy-effective_tld_names.dat` and can be updated following the instructions at Managing advanced configuration.

Note: The `util.httpClientv2.disablePublicSuffixVerification` property defaults to `false` for every host (*=false).

Data type: String

Example: `*=false`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

util.httpClientv2.disableRedirectHandling

Specifies whether or not the HTTP Client automatically handles redirects.

Note: The `util.httpClientv2.disableRedirectHandling` property defaults to `false` for every host (*=false).

Data type: String

Example: `*=false`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

`util.httpClientv2.maxConnections`

Specifies the maximum number of connections that are created in each connection pool.

Note:

- There is a separate connection pool that is created for each unique SSL connection key. This key is generated by using the URL hostname and port, truststore, client keystore, client key alias, protocol, and proxy server values that are specified in the HTTP Client V2 usage.
- The `util.httpClientv2.maxConnections` property defaults to 200 for every host (*=200).

Data type: String

Example: `*=200`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

`util.httpClientv2.maxRouteConnections`

Specifies the maximum number of connections in a connection pool that are available for each unique route.

Note:

The `util.httpClientv2.maxRouteConnections` property defaults to 20 for every host (*=20).

Data type: String

Example: `*=20`

You can specify the value by using one of the following methods:

- Specify a value that applies to every host and port.

```
“*=<value>”
```

- Specify a value on a per host and port basis

```
“<host1>:<port1>=<value>,<host2>:<port2>=<value2>,*=<value3>
```

`util.httpClientv2.proxyHost`

Specifies the hostname of the proxy server if requests must go through a proxy.

To disable the use of a proxy, leave this value, proxyPort and/or proxyProtocol empty.

Note: The `util.httpClientv2.proxyHost` defaults to none.

Data type: `String`

Example: `test.com`

`util.httpClientv2.proxyPort`

Specifies the port of the proxy server if requests must go through a proxy.

To disable the use of a proxy, leave this value, proxyHost and/or proxyProtocol empty.

Note: The `util.httpClientv2.proxyPort` property defaults to none.

Data type: Integer

Example: 443

`util.httpClientv2.proxyProtocol`
Specifies the protocol for the proxy server if requests must go through a proxy.

To disable the use of a proxy, leave this value, proxyHost and/or proxyPort empty.

Note: The `util.httpClientv2.proxyProtocol` property defaults to none.

Data type: `String`

Example: `test.com`

Demo

`live.demos.enabled`
Enables the mobile demonstration application.

Data type: Boolean

Example: `False`

`live.demos.settings`
This setting can be used to pre-populate the settings of the mobile demo. This is a comma separated set of key, value pairs that match what is submitted on the settings form.

Data type: `String`

Example: `lmiHostAndPort=lmi.host.com, lmiAdminId=admin, lmiAdminPwd=admin, acHostAndPort=127.0.0.1, websealHostNameAndPort=webseal.host.com`

Knowledge questions properties

`knowledge.questions.AnswerValidationRegEx`
Specifies the regular expression used to validate the knowledge question answer value provided during a knowledge question management operation. The assigned value is the list of invalid characters to match against to determine if the supplied value is valid.

Note: At a minimum, this property must include the following characters: `<>:"`

Data type: RegEx

Example: `[\[()<>,;:\\/\"\]=]`

`knowledge.questions.QuestionValidationRegEx`
Specifies the regular expression used to validate the knowledge question text value provided during a knowledge question management operation. The assigned value is the list of invalid characters to match against to determine if the supplied value is valid.

Note: At a minimum, this property must include the following characters: `<>:"`

Data type: RegEx

Example: `[\[()<>,;:\\/\"\]=]`

Key encryption and signing service (KESS)

`kess.crlEnabled`
Checks the certificate revocation list. Checking is done by the key encryption and signature service (KESS) for all functions that use an external certificate, except for the audit syslog. If your configuration does not require CRL checking, you can disable it. For example, if you use if an internal certificate authority (CA), you might want to disable CRL checking. The `kess.crlEnabled` property defaults to `true`.

CRL site unavailability scenario
If you have `kess.crlEnabled` set to `true` and a CRL site becomes unavailable, you cannot determine the revocation status of the certificate. In this situation, the single sign-on flow will fail.

Confirm a CRL site unavailability issue by looking for the message "`FBTKJK056E The CRL site could not be determined.`" in the runtime `trace.log` file.

As a temporary workaround, set the CRL checking to `false` to keep the single sign-on flow running. As soon as the CRL site is working again, set `kess.crlEnabled` to `true` so that the single sign-on flow contains the CRL check.

CAUTION: If you do stop CRL checking as a temporary workaround, be aware that the certificate might have already been revoked by the CA. If this type of certificate is allowed to pass the validation, it creates security issues. Therefore, ensure that you enable CRL checking to avoid potential security issues such as this.

Data type: Boolean

Example: `true`

`kess.crlInterval`

The amount of time, in seconds, between successive CRL checks. Using an interval of time between CRL checks reduces the performance impact of doing the checks every time a certificate needs to be validated.

A value less than or equal to zero means that the runtime performs a CRL check every time it wants to use a certificate. The default is `0` seconds.

If `kess.crlEnabled` is set to `false`, this value is ignored.

Data type: Integer

Example: `86400`

This value means that a CRL check on a certificate is performed once per day.

`kess.hostnameValidationDisabled`

Determine whether to disable host name verification when establishing an SSL connection. Host name verification is performed when the host name of the server does not match the CN of the certificate of the server.

In a test environment, you might want to disable the validation. In a production environment, you might want to enable validation.

The default value is `False`.

Data type: Boolean

Example: `False`

`kess.keySelectionCriteria`

Specify which key or certificate to use for signing, validating, encrypting, or decrypting various messages. If there are multiple keys or certificates with the same Subject DN as the key or certificate with the specified alias, this setting determines which one to use. Use one of the following selection methods:

only.alias

Alias only: The selected key only, without Auto rollover. If the key is invalid, the software indicates failure. Configure the property to use the value.

shortest.lifetime

Shortest lifetime: For signing, a valid key with the shortest available lifetime. For validation, key lifetime availability runs from shortest to longest.

longest.lifetime

Longest Lifetime: For signing, a valid key with the longest available lifetime. For validation, key lifetime availability runs from longest to shortest.

Data type: String

Example: `only.alias`

`kessjksservice.exclude.inclusive.namespace.prefixes`

Specifies a comma-separated list of prefix names. When this is set, the prefixes in the list are not added to the InclusiveNamespaces list that is in the Signature Element.

Data type: String

Example: ds

JSON Web Key

`jwks.encryption.keystore`

Defines the name of the encryption keystore to be used by the jwks endpoint for the runtime. These certificates will have their public keys exposed, with the 'use' value 'enc'.

Default value: `rt_profile_keys`

`jwks.signing.keystore`

Defines the name of the signing keystore to be used by the jwks endpoint for the runtime. These certificates will have their public keys exposed, with the 'use' value 'sig'.

Default value: `rt_profile_keys`

Policy information point (PIP)

`pip.uncachedAttributes`

Defines a comma-separated list of attributes that are generated by a policy information point (PIP) that you do not want to be cached.

Data type: String list

Example: `urn:ibm:security:jdbc:city, urn:ibm:security:ldap:priviledgeUser`

Security token service (STS)

`sts.ivcred.unauthenticated.user.name`

Set to a special user account for unauthenticated user tokens when using IVCRED STS module in `validate` mode. The Default value is "".

Data type: `String`

Example: `guest`

`sts.ivcred.unauthenticated.user.registry.id`

In addition to the user name set in `sts.ivcred.unauthenticated.user.name`, a user's registry id can also be added when using IVCRED STS module in `validate` mode. The Default value is "".

This parameter is optional.

Data type: `String`

Example: `cn=guest,o=ibm,c=us`

`sts.ivcred.unauthenticated.user.uuid`

In addition to the user name set in `sts.ivcred.unauthenticated.user.name`, a user's UUID can also be added when using IVCRED STS module in `validate` mode. The Default value is "".

This parameter is optional.

Data type: `String`

Example: `81a2a65e-0018-0150-8080-3f83b0f74f4c`

`sts.ldapAttributeCache.TTL`

Specifies a time-to-live (TTL) value, in seconds, for the amount of time to keep an LDAP attribute in the cache. Specify `0` to disable.

The default value is `60`.

Data type: Integer

Example: `60`

`sts.wstrust.error.shortexception`

Set this parameter to `True` to provide a short exception in the '`wst:Reason`' for STS exceptions. When this parameter is set to `False`, the entire exception stack is provided in '`wst:Reason`'.

Type: Boolean

Default: `False`

Example: `False`

Mobile Multi-Factor Authentication (MMFA)

`mmfa.authenticator.cleanupWait`

The amount of time, in seconds, to wait before another cleanup of expired authenticators is performed.

MMFA authenticator clean up can be disabled by setting `cleanupWait` to `0`.

The default value is `3600`.

Data type: Integer

Example: `3600`

`mmfa.transactionArchival.maxCompletedPerUser`

The number of historical transactions in a completed state to keep in the HVDB before archival to the audit log. The oldest transactions will be removed first. A value of `-1` will indicate that no archival should be performed.

The default value is `50`.

Data type: Integer

Example: `50`

`mmfa.transactionArchival.maxPendingPerUser`

The number of transactions to keep in a pending state. Transactions over this number will have their status set to "fail". The oldest transactions will be aborted first. A value of `-1` will indicate that no archival should be performed.

The default value is `1`.

Data type: Integer

Example: `1`

`mmfa.transactionPending.minAgeBeforeAbort`

The minimum number of seconds a transaction is in the pending state before being aborted via a cleanup thread. Due to the cleanup thread interval, the total time a transaction can be in the pending state can be between `minAgeBeforeAbort` and (`minAgeBeforeAbort` + `cleanupInterval`) - `1`

The default value is `300`.

Data type: Integer

Example: `300`

`mmfa.transactionPending.cleanupInterval`

The number of seconds between each run of the pending transactions cleanup thread.

The default value is `150`.

Data type: Integer

Example: `150`

`mmfa.transaction.cleanupOnlyOnPrimaryMaster`

Indicates whether transaction cleanup should be run on all nodes in a cluster, or only on the primary master. This applies to pending transaction cleanup as well as transaction archival.

The default value is `false`.

Data type: Boolean

Example: `false`

`mmfa.devicePrompt.skipIfOneDevice`

Indicates whether to skip the device selection page in an MMFA flow if the user only has one device or authenticator registered.

The default value is `false`.

Data type: Boolean

Example: `true`

WS-Federation

`wsfed.idp.rstr.excluded.elements`

Specifies a comma-separated list of elements to exclude from the WS-Federation request security token response. Can optionally contain a federation realm and federation partner realm, to indicate the federation or federation partner that uses the property values.

The default value is `default=Forwardable,Delegatable,Status,Renewing`.

The syntax for specifying federation and federation partner is:

```
default=<comma_separated_list_of_elements>:<federation_realm>=<comma_separated_list_of_elements>:
          <federation_realm>%<partner_realm>=<comma_separated_list_of_elements>
```

Data type: String

Example:

```
default=Forwardable,Delegatable,Status,Renewing:fed1-REALM=Forwardable,Delegatable:
fed1-REALM%partner1-REALM=Status
```

SAML 1.1

saml.use.legacy.clockskew.default

IBM Security Verify Access can add a clock skew of 60 seconds when validating the SAML assertion timestamps. To enable the 60 second clock skew, add the custom property:

`saml.use.legacy.clockskew.default = true`

Default value = False

- Value type: Boolean
- Example value: True

Note: This custom property is also applicable for SAML 2.0

saml.allowDebugMessages

When specified as true, and a SAML artifact resolution failure occurs, the SystemOut.log and SystemErr.log contains an informational message. In addition, the message contains extra debug information about the request that contained the failed artifact and provides a reason for the event.

Note: This message is only available in English.

Default value: False

- Value type: Boolean
- Example value: SAML.allowDebugMessage = True

saml.allowNoRecipient

Use this custom property if a SAML 1.x service provider needs to accept a samlp:Response that does not contain a Recipient attribute.

Default value: False

saml.assertion.IncludeNSPrefixList.DS

When this custom property is specified as true, ds is included in the Prefix List attribute of the InclusiveNameSpaces in the SAML assertion.

Default value: False

- Value type: Boolean
- Example value: True

Note: This custom property is also applicable for SAML 2.0

saml.allowSpecificInvalidArtifactMessages

When this custom property is specified as true, and a SAML artifact resolution failure occurs, *identity provider sends a SAML Response with specific invalid message to tell the service provider that there is no assertion available. The specific invalid message is* **FBTSML276E**. If not specified, by default it is false, and the invalid message send back to service provider is **FBTSML013E**.

Default value: False

- Value type: Boolean
- Example value: True

SAML 2.0

`saml20.enableSubjectInAuthnRequest`

Set to `true` if the Subject element is required for the SAML 2.0 AuthnRequest. The Subject element is set to the userid of the existing authenticated session. The Default value is `false`.

Data type: `Boolean`

Example: `true`

`saml20.idp.acsurlpattern`

IBM Security Verify Access uses an exact string comparison between the AssertionConsumerService URL in the AuthnRequest message and the protocol endpoint specified in metadata.

This custom property allows regular expression matching for the AssertionConsumerService URL and the protocol endpoint, so that a dynamic AssertionConsumerService URL that matches the regular expression can be provided in the AuthnRequest.

Data type: String

Note: The binding can be omitted if the configuration applies to all the bindings for that specific federation and partner.

Format:

```
<FederationId>%<PartnerId>
%<Binding>=<RegularExpression>,<FederationId2>%<PartnerId2>
=<RegularExpression2>
```

Example:

```
https://www.myidp.ibm.com/isam/sps/saml20idp/saml20%https://www.mysp.ibm.com
```

```
/isam/sps/saml20sp/saml20%urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST=https://.*.ibm.com/
isam/sps/.*
```

`saml20.sessionStore`

Specifies the SAML 2.0 session footprint store.

The SAML 2.0 is stored in the HVDB by default. When the option is switched to DSC the SAML 2.0 session gets stored in Distributed Session Cache. The Distributed Session Cache (DSC) is an independent service that acts as a centralized session repository for a clustered server environment. Servers in the cluster can use the DSC to provide failover for sessions

DSC as session storage helps to remove the dependency on HVDB for federated single sign-on by using SAML 2.0 protocol (except Alias Service).

Data type: String

Note: The selection for the SAML 2.0 session footprint store is drop-down list with the following options:

- HVDB
- DSC

Example: HVDB

Note: The default value is HVDB.

`saml20.authn.request.provider.name.enabled`

Set to true to add ProviderName value to SAML2.0 AuthnRequests.

Data type: Boolean

Example: False

Note: The default value is `False`.

OIDC

`oidc.rp.idToken.validationSkew`

The number of seconds of skew allowed on the 'nbf' and 'exp' claims of an idToken when it is being processed by an OpenID Connect relying party. For instances where the clocks of two systems are not perfectly synchronized.

Note: This advanced configuration does not apply to legacy OpenID Connect relying parties or Reverse Proxy Relying parties.

Default value: 0

Rhino Javascript Engine

js.optimizationLevel

```
*js.version *Supported values Context.VERSION_ES6, Context.VERSION_1_7, Context.VERSION_1_8
```

This is the rhino javascript version indicator.

Default values: `js.optimizationLevel =0` and `js.version= Context.VERSION_ES6`

Managing user registries

The appliance runtime profile has a user registry associated. Use the User Registry management page to administer the users and group memberships. The user registry in discussion here is the one used by the runtime applications, not the one used by the management interface.

Before you begin

Note: From version 9.0.7 and above, these characters "&|\><;" are not allowed for passwords in the AAC or Federation user registry.

Procedure

1. From the top menu, select the user interface panel for your licensing level.
   - **AAC** > **Manage** > **User Registry**
   - **Federation** > **Manage** > **User Registry**

   A list of all the current users in the registry is displayed. You can filter and reorder the list of users.
2. Select **Users** (current page) or **Groups** to manage users or groups, respectively.

3. To manage users, perform one or more of the following actions as needed:

   Create a user in the registry

   a. Click **New**.
   b. In the **Create User** window, enter the user name and password for the new user.
   c. Click **OK**.

   Delete a user from the registry

   a. Select the user to delete.
   b. Click **Delete**.
   c. In the **Delete User** window, click **Yes** to confirm the delete operation.

   Change the password of a user in the registry

   a. Select the user for which you want to change password.
   b. Click **Set Password**.
   c. In the **Set Password** window, enter the password in the **New Password** and **Confirm Password** fields.
   d. Click **OK**.

   Manage group memberships of a user

   a. Select the user of interest. The group memberships that are associated with this user are displayed under the **Group Membership** section.
   b. You can add the user to a group or delete the user from a group in the registry.

      Add the user to a group

      1) In the **Group Membership** section, click **Add**.
      2) In the **Add to Group** window, select the group to add this user to.

         Note: Only a single group can be selected.
      3) Click **OK**.

      Remove the user from a group

      1) In the **Group Membership** section, select the group to remove the user from.
      2) Click **Delete**.
      3) In the **Remove from Group** window, click **Yes** to confirm the removal.

4. To manage groups, perform one or more of the following actions as needed:

   Create a new group in the registry

   a. Click **New**.
   b. In the **New Group** window, enter the group name for the new group.
   c. Click **OK**.

   Delete a group from the registry

   a. Select the group to delete.
   b. Click **Delete**.
   c. In the **Delete Group** window, click **Yes** to confirm the delete operation.

   Manage group members

   a. Select the group of interest. The users that are currently members of this group are displayed under the **Group Members** section.
   b. You can add a user to the group or delete a user from the group in the registry.

      Add a user to the group

      a. In the **Group Members** section, click **Add**.
      b. In the **Add to Group** window, select the user to add to the group.

         Note: Only a single user can be selected.
      c. Click **OK**.

      Remove a user from the group

      a. In the **Group Members** section, select the user to remove from the group.
      b. Click **Remove**.
      c. In the **Remove from Group** window, click **Yes** to confirm the removal.

Tuning runtime application parameters and tracing specifications

To manually tune selected runtime application parameters and tracing specifications, use the **Runtime Parameters** management page.

Before you begin

About this task

Procedure

1. From the top menu, select **AAC** > **Global Settings** > **Runtime Parameters** or **Federation** > **Global Settings** > **Runtime Parameters**.

   This page contains three panels: **Runtime Status**, **Runtime Tuning Parameters**, and **Runtime Tracing**.

2. Perform one or more of the following actions to tune your runtime.

   Note: Certain changes might require a restart of the runtime before they can take effect.

   Disable automatic restart of the runtime

   By default, the runtime is automatically restarted after certain changes are made. You can disable this automatic restart function if you prefer manual restarts.

   a. On the **Runtime Tuning Parameters** panel, select **Auto Restart**.
   b. Click **Edit**.
   c. In the **Auto Restart** window, define the value as **False**.
   d. Click **OK**.

   View the status of the runtime and restart the runtime

   a. Select the **Runtime Status** panel. The status of local and clustered runtimes are displayed.
      - Under **Local Runtime Status**, you can view the runtime operational status, when it was last started, and whether a restart is outstanding. If the value of the **Restart Required** field is **True**, it means that the runtime must be restarted for some changes to take effect.
      - Under **Clustered Runtime Status**, all nodes in the cluster are listed.
        – The **Master** column indicates whether a node is the cluster master.
        – The **Runtime Status** column indicates whether a node is running or stopped.
        – The **Changes Active** column indicates whether changes made to the cluster configuration are active on this node. Having a green indicator in this column means that all changes

made are already active. Having a yellow indicator in this column means that this node must be restarted before some changes can take effect.

b. Depending on which runtime you want to restart, click **Restart Local Runtime** or **Restart All Clustered Runtimes**.

Modify the maximum or initial heap size

These parameters indicate the maximum and initial heap size in megabytes for the runtime Java virtual machine.

a. On the **Runtime Tuning Parameters** panel, select **Max Heap Size** or **Initial Heap Size**.
b. Click **Edit**.
c. In the **Max Heap Size** or **Initial Heap Size** window, enter the heap size value as needed.
d. Click **OK**.

Modify the minimum or maximum threads

These parameters indicate the minimum number of core threads that the runtime server starts with and the maximum number of threads that can be associated with the runtime server.

If the minimum value is not set or is set as `-1`, a default value is calculated based on the number of hardware threads on the system.

If the maximum value is not set or is set as 0 or less, a default value of unbounded is used.

The minimum **cannot** be set to a value larger than the maximum.

a. On the **Runtime Tuning Parameters** panel, select **Min Threads** or **Max Threads**.
b. Click **Edit**.
c. In the **Min Threads** or **Max Threads** window, enter the required value.
d. Click **OK**.

Modify whether to suppress sensitive trace

Enabling this parameter prevents sensitive information from being exposed in log and trace files. Examples of such sensitive information include bytes received over a network connection.

a. On the **Runtime Tuning Parameters** panel, select **Suppress Sensitive Trace**.
b. Click **Edit**.
c. In the **Suppress Sensitive Trace** window, select or clear the check box as needed.
d. Click **OK**.

Modify console log level

Console log level controls the granularity of messages that go to the `console.log` file.

a. On the **Runtime Tuning Parameters** panel, select **Console Log Level**.
b. Click **Edit**.
c. In the **Console Log Level** window, select the new value from the list.
d. Click **OK**.

Set whether to accept client certificates

This parameter controls whether the server accepts client certificates as a form of authentication.

a. On the **Runtime Tuning Parameters** panel, select **Accept Client Certificates**.
b. Click **Edit**.
c. In the **Accept Client Certificates** window, select or clear the check box as needed.
d. Click **OK**.

Maximum Session Count

This parameter defines the maximum number of sessions that is maintained in memory.

Note: The default setting is 250000. When this setting is used, the maximum number of sessions is 250000.

a. On the **Runtime Tuning Parameters** panel, select **Maximum Session Count**.
b. Click **Edit**.
c. In the **Maximum Session Count** window, define the value.
d. Click **OK**.

Set session invalidation timeout

This parameter defines the amount of time a session can remain unused before it is no longer valid.

Note: The default setting is `1200`. When this setting is used, the session invalidation timeout is 1200 seconds.

a. On the **Runtime Tuning Parameters** panel, select **Session Invalidation Timeout**.
b. Click **Edit**.
c. In the **Session Invalidation Timeout** window, define the value in seconds.
d. Click **OK**.

Set session reaper poll interval

This parameter defines the wake-up interval in seconds for the process that removes invalid sessions. The minimum value is 30 seconds.

The default setting is **Unset**. When this setting is used, or if a value less than the minimum is entered, an appropriate value is automatically determined and used. This value overrides the default installation value, which is 30 - 360 seconds, based on the session invalidation timeout value. Because the default session invalidation timeout is 1800 seconds, the reaper interval is usually between 120 and 180 seconds.

a. On the **Runtime Tuning Parameters** panel, select **Session Reaper Poll Interval**.
b. Click **Edit**.
c. In the **Session Reaper Poll Interval** window, define the value in seconds.
d. Click **OK**.

Set the keystore that is used by the runtime server

This parameter defines the key database that contains the runtime server's private key.

a. On the **Runtime Tuning Parameters** panel, select **Keystore**.
b. Click **Edit**.
c. In the **Keystore** window, select the key database from the list.
d. Click **OK**.

Set the truststore that is used by the runtime server

This parameter defines the key database that contains keys that are trusted by the runtime server

a. On the **Runtime Tuning Parameters** panel, select **Truststore**.
b. Click **Edit**.
c. In the **Truststore** window, select the key database from the list.
d. Click **OK**.

Configure an outbound HTTP proxy

You must specify values for the properties for the HTTP proxy. You might also need to import the root CA certificate from the proxy. See the instructions that follow.

Table 102. HTTP proxy properties

Name	**Sample Value**	**Description**
`http.proxyHost`	`http.proxy.ibm.com`	The hostname or IP address of the HTTP proxy
`http.proxyPort`	3128	The port of the HTTP proxy
`https.proxyHost`	`https.proxy.ibm.com`	The hostname or IP address of the HTTPS proxy

Table 102. HTTP proxy properties (continued)

Name	Sample Value	Description
`https.proxyPort`	3128	The port of the HTTPS proxy

a. For each property in the table above:

   1) On the **Runtime Tuning Parameters** panel, select the property.

   2) Click **Edit**.

   3) In the property window, enter the value. See the sample values in the table.

   4) Click **OK**.

b. When all properties are set, follow the prompt to deploy the pending changes.

Certain functions, such as the OpenID connect single sign-on flow, require the root CA certificate of the outbound HTTP proxy to be imported to the Security Verify Access runtime keystore.

Complete the following steps:

a. Go to your HTTP Proxy application and obtain the necessary certificate for exchange. The exact steps to take are specific to the proxy application. Place the certificate on the local file system where it can be accessed by the appliance.

b. On the Security Verify Access system, log in to the local management interface and select **System** > **Secure Settings** > **SSL Certificates**

c. Select the `rt_profile_keys` keystore.

d. Select **Manage** > **Edit SSL Certificate Database**.

e. Select **Manage** > **Import**.

f. On the Signer Certificate panel, browse to locate the **Certificate File**. Enter a **Certificate Label**. Click **Import**.

g. Deploy the changes.

Delete the value of a parameter

Use this button to delete the existing value of a parameter.

a. Select the parameter to reset the value for.

b. Click **Delete**. The value of the parameter is then changed to `Unset`.

Manage the application interface on which the runtime listens

a. On the **Runtime Tuning Parameters** panel, under **Runtime Listening Interfaces**, you can add, edit, or delete a listening interface.

   To add a listening interface

   1) Click **Add**.

   2) In the **Runtime Listening Interfaces** window, select the listening interface from the list.

   3) Specify the listening port.

   4) Select the **SSL** check box if security is required.

   5) Click **OK**.

   To modify a listening interface

   1) Select the listening interface to edit.

   2) Click **Edit**.

   3) In the **Runtime Listening Interfaces** window, edit the values as needed.

   4) Click **OK** to save the changes.

To delete a listening interface

1) Select the listening interface to delete.
2) Select **Delete**.
3) Confirm the deletion.

Manage tracing specification

Note: Setting trace for Oracle components "`oracle.*`" results in the underlying Oracle JDBC jar file being changed to a debugging jar file. This might have adverse effects on performance and as such Oracle tracing should only be enabled for debugging purposes and disabled once complete.

a. Select the **Runtime Tracing** link from the top of this page. You can also access this panel from the top menu by selecting **Monitor** > **Logs** > **Runtime Tracing**.
b. Use one of the following ways to edit the trace level of a component.
   - Select the component name from the **Component** list. Select the ideal trace level for this component from the **Trace Level** list. Then, click **Add**. Repeat this process to modify trace levels for other components if needed. To clear all of the tracing levels, click **Clear**.

     To log all events, select ALL as the trace level.

     Note: This setting increases the amount of data in logs, so use this level when necessary.

```
com.tivoli.am.fim.authsvc.*
com.tivoli.am.fim.trustserver.sts.modules.*
```

Table 103. Valid trace levels. The following table contains the valid trace levels.

Level	**Significance**
ALL	All events are logged. If you create custom levels, ALL includes those levels and can provide a more detailed trace than FINEST.
FINEST	Detailed trace information that includes all of the details that are necessary to debug problems.
FINER	Detailed trace information.
FINE	General trace information that includes methods entry, exit, and return values.
DETAIL	General information that details sub task progress.
CONFIG	Configuration change or status.
INFO	General information that outlines the overall task progress.
AUDIT	Significant event that affects the server state or resources.
WARNING	Potential error or impending error. This level can also indicate a progressive failure. For example: the potential leaking of resources
SEVERE	The task cannot continue, but component, application, and server can still function. This level can also indicate an impending unrecoverable error.

Table 103. Valid trace levels. The following table contains the valid trace levels. *(continued)*

Level	Significance
FATAL	The task cannot continue, and component, application, and server cannot function.
OFF	Logging is turned off.

- Enter the name and value of the trace component in the **Trace Specification** field. To modify multiple components, separate two strings with a colon (:). Here is an example.

```
com.x.y.*=WARNING:com.a.b.*=WARNING:com.ibm.isva.*=INFO
```

c. Click **Save**.

3. When you make changes, the appliance displays a message that there are undeployed changes. If you have finished making changes, deploy them.

Template files

Template files are HTML pages that are presented to your users during the authentication process. The pages prompt users for authentication information, such as user names and passwords, or present information to users, such as one-time passwords, status, or errors.

You can customize any of the HTML pages by exporting, modifying, and importing its corresponding template file. Each template file uses one or more specific macros.

Managing template files

Use the local management interface to manage files and directories in the template files.

About this task

You can run the following tasks on the template files:

- **New**- Use this option if you want to create a new file or directory.
- **Edit**- Use this option if you want to view or modify the template file.
- **Import**- Use this option if you to import a file to the template files root.
- **Export**- Use this option if you want to export a file from the template files root.
- **Rename**- Use this option if you want to rename a file or directory from the template files root.
- **Delete**- Use this option if you want to delete a file or directory from the template files root.
- **Import Zip**- Use this option if you want to import the template files from a compressed file.
- **Export Zip**- Use this option if you want to export the template files as a compressed file.

  Note: When you use this option to export template files as a compressed file, you cannot export an individual folder. The root directory, including all the sub-directories, is exported. To access the contents of a specific directory, export the entire template directory, and then view the directory from the extracted compressed file on your local workstation. Administrators can refer to `metadata.json` under file downloads after upgrades to check if there are new configuration parameters included for AAC related endpoints.

Procedure

1. Select **AAC** > **Global Settings** > **Template Files**

2. Work with all the management files and directories.

   Create a file or directory in the template files root

   a. Select the directory of interest.
   b. Select **New**.
   c. Select **File** or **Directory**.
   d. Enter the name of the file or directory.
   e. Click **Save**.

   View or update the contents of a file in the template files root

   a. Select the file of interest.
   b. Select **Edit**. You can then view the contents of the file.
   c. Edit the contents of the file.
   d. Click **Save**.

   Export a file from the template files root

   a. Select the file of interest.
   b. Select **Manage** > **Export**.
   c. Confirm the save operation when your browser displays a confirmation window.

   Rename file from the template files root

   a. Select the file or directory of interest.
   b. Select **Manage** > **Rename**.
   c. Enter the new resource name.
   d. Click **Save**.

   Delete file from the template files root

   a. Select the file or directory of interest.
   b. Select **Manage** > **Delete**.
   c. Click **Yes**.

   Import a file to the template files root

   - Select a file.
     a. Select **Manage** > **Import**.
     b. Click **Browse**.
     c. Browse to the file that you want to import the contents from.
     d. Click **Open**.
     e. Click **Import**.
   - Select a folder.
     a. Select **Manage** > **Import**.
     b. Click **Browse**.
     c. Browse to the file that you want to import to the selected folder.
     d. Click **Open**.
     e. Click **Import**.

   Export the template file as a compressed file

   a. Select **Manage** > **Export Zip**.
   b. Confirm the save operation when your browser displays a confirmation window.

Import the template files as a compressed file

Make sure that the files in the compressed file are in the same directory structure as the files in the root directory or appliance.

For example, if a file in the compressed file is in the `/C` directory of the appliance, the compressed file must contain the C folder and the file that you want to import. When you import a compressed file that contains:

- A file that exists in the appliance

  The file is replaced in the appliance.
- A file or directory that does not exist in the appliance

  The file or directory is created in the appliance. You can put these new files and directories in an existing non-root directory or add a new directory in the root.

  Note: You cannot delete a top level directory after you create it.

a. Select **Manage** > **Import Zip**.
b. Click **Browse**.
c. Browse to the file you want to import.
d. Click **Open**.
e. Click **Import**.

3. When you edit or import template files, the appliance displays a message that there are undeployed changes. If you finish the changes, deploy them.

   For more information, see Deploying pending changes.

Customizing the consent page

The consent page of an OpenID Connect Provider Federation can be changed with the Template Files page in the local management interface.

About this task

All OpenID Connect Provider (OP) federations can have their own unique consent pages. Follow these steps to set a consent page to be used by a specific federation.

Procedure

1. Log in to the local management console.
2. Select **Federation** > **Global Settings** > **Template Files**.
3. Expand the **C** locale.
4. Highlight the **oidc** folder.
5. Click **New** and select **Directory**.
6. Enter the **Federation Name** of the OpenID Connect Provider Federation to use the custom consent page.
7. Click **Save**.
8. Highlight the new directory.
9. Click **New** and select **File**.
10. Enter `consent.html` as the file name.
11. Populate the file contents.
12. Click **Save**.
13. Deploy the pending changes.

    Note: The deploy operation triggers a runtime restart.

Template page scripting

You can use JavaScript to add server-side scripting for Advanced Access Control and Federation template pages. You can use JavaScript functions, closures, objects, and delegations.

Usage

You can customize template files or pages on the server. For example, you can customize an error message that is displayed by the runtime server.

The template files menu is located under both the Federation and AAC menus.

To edit a Federation template file, go **Federation > Template Files**, select the specific template file, and click **Edit**.

To edit an AAC template file, go to **AAC > Template Files**, select the specific template file, and click **Edit**.

The JavaScript engine supports the following syntax:

- Insert JavaScript code between <% and %>.
- Embed JavaScript expressions between <%= and %>.

Example tasks

- Access whitelisted Java classes. For example,

  ```
  var javaStr = new java.lang.String("Hello")
  ```

- Access all the macro variables through templateContext. The standard object to access a Java object is templateContext. For example,

  ```
  templateContext.macros["@TIMESTAMP@"]
  ```

- Use the document.write function to write content to the output stream. For example,

  ```
  templateContext.response.body.write("Hello")
  ```

Examples

Table 104. Example JavaScript

Template HTML	**Output**
`<% var contents = {product:"Verify Access",department:"Lab",country:"SG",region:"Asia"}; templateContext.response.body.write(contents.product); %>`	Verify Access
`<% var date = templateContext.macros["@TIMESTAMP@"].substring(0, 10); templateContext.response.body.write(date); %>`	2017-01-25

The following code example shows how to use repeatable macros. The following example shows an OAuth consent page.

```
<%
var test = templateContext.macros["oauthTokenScopeNewApprovalRepeatable"];
n = test.length;
for (i=0; i<n; i++){
  var scope = test[i]["@OAUTH_TOKEN_SCOPE_REPEAT@"];
  if (scope == "contacts"){
   label ="Do you grant permission to the client to access your phone book";
  }
```

```
   else if (scope == "photos"){
    label ="Do you grant permission to the client to access your photos";
   }
   else if (scope == "messages"){
    label ="Do you grant permission to the client to access your WhatsApp messages";
   }
   else{
    label ="Do you grant permission to the client to access your "+scope;
   }
%>
```

Setting an HTTP response header

You can use `templateContext.response.setHeader(HeaderName, HeaderValue)` to set an HTTP response header.

For example, you can set the `Content-Type` to support both a mobile-based browser and a traditional browser. A mobile-based browser might expect JSON format while a traditional browser expects forms-based HTML.

```
 <%
templateContext.response.setHeader("Content-Type","application/json");
var myObj = { "name":"John", "age":31, "city":"New York" };
templateContext.response.body.write(JSON.stringify(myObj));
%>
```

To set an HTTP header that uses forms-based HTML:

```
templateContext.response.setHeader("Content-Type","text/html");
```

Setting an HTTP status code

You can use `templateContext.response.setStatus(Code)` to set an HTTP response status code.

For example, if you want to set the status to `400` (standard code for a bad request):

```
templateContext.response.setStatus(400);
```

Setting a Redirect URL

You can use `templateContext.response.sendRedirect(URL)` to redirect the HTTP response to a different URL.

For example, when you configure single logout, you can redirect the response to a specific target page, based on the federation name. An example scenario is a deployment that has one SAML 2.0 federation with two partner federations. The partner federations are named `saml20app2` and `saml20sp`. The `saml20app2` federation uses an application that is named `jkebank`. The `saml20sp` federation uses an application that is named `jkeschool`. The page to display on logout is determined by the federation name.

```
var fedName = templateContext.macros[@FEDERATION_NAME@"];
if (fedName == "saml20app2")
{
    templateContext.response.sendRedirect("http://jkebank:1337");
}
else if
{
(fedName == "saml20sp")
{
    templateContext.response.sendRedirect("http://jkeschool:1400");
}
```

Obtaining a list of macros that are available for a template page

In some scenarios, you might want to write JavaScript based on configuration values in your deployment. For example, you might implement one action based on the authentication type, such as if the OTP type is TOTP. Another example is you might implement an action if the Federation name of the single sign-on partner matches a certain value.

Information such as the OTP type and partner name can be retrieved only through the template page macros. To use such information, you need to know which macros are used by the page. The JavaScript engine support provides a utility that can print the available macros for a page.

Use the following syntax to obtain a list of the available macros.

```
<% templateContext.response.body.write(JSON.stringify(templateContext.macros)); %>
```

For example, the following sample code prints the macros from a template page that ran a single sign-on flow with a partner that does not exist.

```
{

    "@PAGE_IDENTIFIER@": "/saml20/invalid_init_msg.html",
    "@TARGET@": "https://www.mysp.ibm.com/isam/mobile-demo/diag",
    "@PARTNER_ENTITY_ID@": "",
    "@ERROR_MESSAGE@": "FBTSML002E The value https://saml.partner.com for attribute PartnerId is not
 valid.",
    "@FEDERATION_NAME@": "saml20idp",
    "@FEDERATION_ENTITY_ID@": "https://www.myidp.ibm.com/isam/sps/saml20idp/saml20",
    "@REQ_ADDR@": "/sps/saml20idp/saml20/logininitial",
    "@ERROR_CODE@": "FBTSML002E",
    "@EXCEPTION_STACK@": "",
    "@PARTNER_NAME@": "",
    "@TIMESTAMP@": "2017-06-22T03:34:39Z",
    "@SAMLSTATUS@": "<fim:FIMStatusCollection xmlns:fim=\"urn:ibm:names:ITFIM:saml\"
     xmlns:samlp=\"urn:oasis:names:tc:SAML:2.0:protocol\"><fim:FIMStatusCollectionEntry>
     <samlp:Status><samlp:StatusCode Value=\"urn:oasis:names:tc:SAML:2.0:status:Responder\"></
samlp:StatusCode>
     <samlp:StatusDetail><fim:FIMStatusDetail MessageID=\"invalid_attribute_value\">
     <fim:SubstitutionString>https://saml.salesforce.com</fim:SubstitutionString>
     <fim:SubstitutionString>PartnerId</fim:SubstitutionString></fim:FIMStatusDetail>
     </samlp:StatusDetail></samlp:Status></fim:FIMStatusCollectionEntry></fim:FIMStatusCollection>",
    "@EXCEPTION_MSG@": ""

}
```

The format is JSON `{ "name1":"value1","name2":"value2"}`

Limitations

- JavaScript validation is done only when a template file is edited (imported) or created. A template file that is imported as a part of an Import compressed file is not validated.
- You must restart the runtime manually to activate changes to OpenID Connect template files. In the administrative interface, click **Federation -> Runtime Tuning -> Restart Runtime**.
- When you access a variable, do not end the variable name with a semicolon. For example, in the following JavaScript, do not end <%=example%> with a semicolon <%=example;%>.

  ```
  <%var example = "Hello World"; %>
  <%=example%>
  ```

  The correct syntax is <%=example%>. Do not use the incorrect syntax <%=example;%>.

Template files reference

Template files are HTML pages that are presented to your users during the authentication process. The pages prompt users for authentication information, such as user names and passwords, or present information to users, such as one-time passwords, status, or errors.

Consent to register device template files

These files support consent to registering a device.

Consent to register device template files

These files support consent to registering a device.

Table 105. Default template files in the ac/ directory

Page name	File name and macros	Description
Attribute Collection JavaScript	`ac/info.js`	Detects the location of the device from which the requests are made. Collects the web browser attributes and sends them to the server for storing in the database. When the user logs out or when the current session times out, the script deletes the attributes from the database.
Dynamics Attributes JavaScript	`ac/javascript_rules/dynamic_attributes.js`	Runs after each request is processed by risk engine. Use it to capture attributes that do not get captured automatically. Captured attributes are stored either in the session storage or the behavior storage area of the risk-based component, or both. The risk profile configuration dictates where the attributes are stored.

User self-care template files

These files support user self-care tasks.

User self-care template files

These files support user self-care tasks.

Table 106. Default template files in the mga/ directory

Page name	File name and macros	Description
Common User Profile Management JavaScript	`mga/user/mgmt/common.js`	Used by one-time password pages and by device management pages. Contains functions and properties that are used for making calls to the user self-care REST services.
Device Attributes	`mga/user/mgmt/device/device_attributes.html`	Enables or disable devices. Renames or removes device. Displays all of the attributes for a device. For more information, see Managing your registered devices.

Table 106. Default template files in the mga/ directory (continued)

Page name	File name and macros	Description
Device Attributes JavaScript	`mga/user/mgmt/device/device_attributes.js`	Processes values that are entered in the `device_attributes.html` template
Device Selection	`mga/user/mgmt/device/device_selection.html`	Displays device name, status (enabled or disabled), and time of last activity. For more information, see Managing your registered devices.
Device Selection JavaScript	`mga/user/mgmt/device/device_selection.js`	Processes data to display in the `device_selections.html` template
Authorization Grant	`mga/user/mgmt/device/grant_attributes.html`	Enables or disables an OAuth 2.0 authorization grant. Removes an OAuth 2.0 authorization grant. Displays the OAuth 2.0 tokens and attributes of an authorization grant. For more information, see Managing OAuth 2.0 authorization grants.
Authorization Grants JavaScript	`mga/user/mgmt/device/grant-attributes.js`	Processes data to display in the `grant_attributes.html` template.
HMAC OTP Shared Key	`mga/user/mgmt/otp/otp.html`	Resets TOTP and HOTP Secret Key. For more information, see Managing OTP secret keys.
HMAC OTP Shared Key JavaScript	`mga/user/mgmt/otp/otp.js`	Resets TOTP and HOTP Secret Key.
Knowledge Questions management	`mga/user/mgmt/questions/user_questions.html` Macros: • **`@USERNAME @`** • **`@MAX_STORED_QUESTIONS@`**	Displayed for the user to manage their knowledge questions. The user can select pre-configured questions or write their own questions.
Knowledge Questions JavaScript functions	`mga /user/mgmt/questions/user_questions.js`	Consists of the JavaScript functions that: • Display the knowledge questions. • Allow the user to manage their knowledge questions.

Authentication process

These files support the authentication process

Authentication process template files

These files support the authentication process. For more information, see Authentication.

Table 107. Default template files in the `authsvc/` directory

Page name	File name and macros	Description
Server Error	`authsvc/server_error.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays general server errors.
User Error	`authsvc/user_error.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during authentication policy execution that are caused by user input.

Authentication mechanisms

These files support the authentication mechanisms.

Authentication mechanisms

These files support the authentication mechanisms. For more information, see Authentication.

Table 108. Default template files in the `otp/` directory

Page name	File name and macros	Description and link to file contents
Change PIN required	`otp/change_pin.html` Macros: • **`@ERROR_MESSAGE@`** • **`@MAPPING_RULE_DATA@`** • **`@DISPLAY_RESELECT_BUTTON@`**	Enables the user to enter a new PIN.

Table 108. Default template files in the `otp/` directory (continued)

Page name	**File name and macros**	**Description and link to file contents**

OTP Email Delivery Message	`otp/delivery/email_message.xml`	Used by `EmailOTPDelivery` as the content of the email that it sends to the user.
		
		 The template file must be a compliant XML file.
		
		 The content can be plain text or HTML. Following is an example using HTML in the email template:
		
		 `<?xml version="1.0" encoding="UTF-8"?>`
		 `<root>`
		 `<Subject>`
		 `  <Value>One-time Password</Value>`
		 `</Subject>`
		 `<Message>`
		 `  <Value><![CDATA[<html>`
		 `<body>`
		 `<img src="https://www.example.com/images/logo.gif" />`
		 `<br />This is your HTML email one-time password @OTP_STRING@.<br />`
		 `    <p>Thank you,<br />`
		 `    The Example Co.</p>`
		 `</body>`
		 `    </html>]]>`
		 `  </Value>`
		 `</Message>`
		 `</root>`
		
		 For information on HTML formatting of email messages, see HTML format for OTP email messages.

OTP SMS Delivery Message	`otp/delivery/sms_message.xml`	Used by `SMSOTPDelivery` as the content of the SMS that it sends to the user. The template file must be a compliant XML file.
One-Time Password Delivery Selection	`otp/delivery_selection.html` Macros: • **`@OTP_METHOD_CHECKED@`** • **`@OTP_METHOD_LABEL@`**	Displays the list of methods for generating, delivering, and verifying the one-time password.

Table 108. Default template files in the `otp/` directory (continued)

Page name	File name and macros	Description and link to file contents
OTP General Error	`otp/errors/allerror.html` Macros: • **@REQ_ADDR@** • **@TIMESTAMP@** • **@DETAIL@** • **@EXCEPTION_STACK@**	Displays general errors that happen during the one-time password flow.
OTP Validation Error	`otp/errors/error_could_not_validate_otp.html` Macros: • **@REQ_ADDR@** • **@TIMESTAMP@** • **@DETAIL@** • **@EXCEPTION_STACK@**	Displays errors during the validation of the one-time password that the user submits.
OTP Generation Error	`otp/errors/error_generating_otp.html` Macros: • **@REQ_ADDR@** • **@TIMESTAMP@** • **@DETAIL@** • **@EXCEPTION_STACK@**	Displays errors during the generation of a one-time password.
OTP Methods Retrieval Error	`otp/errors/error_get_delivery_options.html` Macros: • **@REQ_ADDR@** • **@TIMESTAMP@** • **@DETAIL@** • **@EXCEPTION_STACK@**	Displays errors during the retrieval of the list of methods for delivering one-time password to the user.
OTP Delivery Error	`otp/errors/error_otp_delivery.html` Macros: • **@REQ_ADDR@** • **@TIMESTAMP@** • **@DETAIL@** • **@EXCEPTION_STACK@**	Displays errors during the delivery of a one-time password to the user.

Table 108. Default template files in the `otp/` directory (continued)

Page name	File name and macros	Description and link to file contents
OTP STS Invocation Error	`otp/errors/error_sts_invoke_failed.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@DETAIL@`** • **`@EXCEPTION_STACK@`**	Displays errors during the invocation of the Security Token Service.
One-Time Password Login	`otp/login.html` Macros: • **`@ERROR_MESSAGE@`** • **`@MAPPING_RULE_DATA@`** • **`@DISPLAY_RESELECT_BUTTON@`**	Displays the form where the user can enter the one-time password.
Enter Next OTP Form	`otp/next_otp.html` Macros: • **`@ERROR_MESSAGE@`** • **`@MAPPING_RULE_DATA@`** • **`@DISPLAY_RESELECT_BUTTON@`**	Enables the user to enter the next one time password.

Table 109. Default template files in the `authsvc/authenticator/password/` directory

Page name	File name and macros	Description
Change Password	`authsvc/authenticator/password/change_password.html` Macros: • **`@USERNAME@`** • **`@ERROR_MESSAGE@`**	Enables the users to change their registry password.
Username and Password Error	`authsvc/authenticator/password/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the user name and password authentication or when the users modify their password.
Username and Password Login	`authsvc/authenticator/password/login.html`	Displays the form where the users can enter their user name and password to log in.

Table 110. Default template files in the `authsvc/authenticator/http_redirect/` directory

Page name	File name and macros	Description
HTTP Redirect Authentication Error	`authsvc/authenticator/http_redirect/allerror.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays general errors during for HTTP redirect authentication mechanism.
HTTP Redirect Authentication Failed	`authsvc/authenticator/http_redirect/error_authenticate.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the HTTP redirect authentication flow.

Table 111. Default template files in the `authsvc/authenticator/macotp/` directory

Page name	File name and macros	Description
MAC One-Time Password Delivery Selection	`authsvc/authenticator/macotp/delivery_selection.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays the list of methods for generating, delivering, and verifying the one-time password.
MAC OTP One-Time Password Error	`authsvc/authenticator/macotp/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the MAC one-time password authentication.

Table 111. Default template files in the `authsvc/authenticator/macotp/` directory (continued)

Page name	File name and macros	Description
MAC One-Time Password Login	`authsvc/authenticator/macotp/login.html` Macros: • **`@OTP_HINT@`** • **`@OTP_LOGIN_DISABLED@`**	Displays the form where the user can enter the MAC one-time password

Table 112. Default template files in the `authsvc/authenticator/rsa/` directory

Page name	File name and macros	Description
RSA One-Time Password Error	`authsvc/authenticator/rsa/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the RSA one-time password authentication.
RSA One-Time Password Login	`authsvc/authenticator/rsa/code.html` Macro: **`@ERROR_MESSAGE@`**	Displays the form where the users can enter the RSA one-time password to log in.
RSA One-Time Password Login (New PIN)	`authsvc/authenticator/rsa/new_pin.html` Macro: **`@ERROR_MESSAGE@`**	Enables users to enter a new RSA pin.
RSA One-Time Password Login (Next OTP)	`authsvc/authenticator/rsa/next_code.html` Macro: **`@ERROR_MESSAGE@`**	Enables users to enter the next RSA one-time password.

Table 113. Default template files in the `authsvc/authenticator/totp/` directory

Page name	File name and macros	Description
TOTP One-Time Password Error	`authsvc/authenticator/totp/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the TOTP one-time password authentication.
TOTP One-Time Password Login	`authsvc/authenticator/totp/login.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays the form where the users can enter the TOTP password to log in.

Table 114. Default template files in the `authsvc/authenticator/hotp/` directory

Page name	File name and macros	Description
HOTP One-Time Password Error	`authsvc/authenticator/hotp/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the HOTP one-time password authentication.
HOTP One-Time Password Login	`authsvc/authenticator/hotp/login.html` Macros: **`@ERROR_MESSAGE@`**	Displays the form where the users can enter the HOTP password to log in.

Table 115. Default template files in the `authsvc/authenticator/consent_register_device/` directory

Page name	File name and macros	Description
Consent to Device Registration Error	`authsvc/authenticator/consent_register_device/error.html` Macros: • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during the consent to device registration flow.
Consent page	`authsvc/authenticator/consent_register_device/consent-form.html` Macro: **`@ERROR_MESSAGE@`**	Prompts the user to provide consent for registering a device.

Table 116. Default template files in the `authsvc/authenticator/eula/` directory

Page name	File name and macros	Description

End-User License Agreement license file display	`authsvc/authenticator/eula/license.txt`	Contains the license agreement to display to the user.
		
		The template does not use replacement macros.
		
		Note: You can add more license files to the template tree.
		
		Specify the metadata in the End-User License Agreement for the following purposes:
		• Auditing
		• Forensic
		
		The End-User License Agreement authentication mechanism removes the metadata before it displays the license agreement to the user. The metadata must be on the first line of the license agreement. For example:
		`Metadata:   Version: 1.0`
		`          Identifier:`
		` 135223434343`
		
		When the user accepts the license agreement or declines the license agreement, the mechanism audits:
		• The user action.
		• The license file name.
		• The corresponding metadata.

End-User License Agreement license agreement display	`authsvc/authenticator/eula/eula.html` Macros: • **`@USERNAME@`** • **`@LICENSE@`**	Displays the page where the user views the license and accepts the license agreement.

Table 116. Default template files in the `authsvc/authenticator/eula/` directory (continued)

Page name	File name and macros	Description
End-User License Agreement license agreement decline	`authsvc/authenticator/eula/error_license_declined.html` Macros: • **`@USERNAME@`** • **`@ERROR_MESSAGE@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@ERROR_MESSAGE@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`** • **`@LICENSE_FILE@`** • **`@LICENSE_METADATA@`**	Displays the page where the user declines the license agreement.

Table 117. Default template files in the `authsvc/authenticator/knowledge_questions/` directory

Page name	File name and macros	Description
Knowledge Questions authentication mechanism knowledge question form	`authsvc/authenticator/knowledge_questions/login.html` Macros: • **`@ QUESTION_TEXT @`** • **`@ QUESTION_INDEX @`** • **`@QUESTION_UNIQUE_ID@`** • **`@QUESTION_COUNT@`** • **`@ERROR_MESSAGE@`** • **`@NUM_REQUIRED_ANSWERS@`**	Displays the form where the user enters the answers to the required knowledge questions.
Knowledge Questions authentication mechanism knowledge question authentication errors	`authsvc/authenticator/knowledge_questions/error.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@ERROR_MESSAGE@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors during knowledge-question authentication.

Table 117. Default template files in the `authsvc/authenticator/knowledge_questions/` directory (continued)

Page name	File name and macros	Description
Knowledge Questions authentication mechanism missing knowledge questions with grace period	`authsvc/authenticator/knowledge_questions/not_enough_questions_found_continue.html` Macros: • **`@USERNAME@`** • **`@NUM_REQUIRED_ANSWERS@`** • **`@NUM_REGISTERED_QUESTIONS@`** • **`@GRACE_PERIOD_AUTH_COUNT@`** • **`@MAX_GRACE_PERIOD_AUTH_COUNT@`**	Displayed when the user did not register the required number of knowledge questions and answers that are required for successful authentication. The following conditions must also be true: • The administrator configured the environment to allow grace-period authentication. • The user did not reach the limit of grace-period logins.
Knowledge Questions authentication mechanism missing knowledge questions without grace period	`authsvc/authenticator/knowledge_questions/not_enough_questions_found_error.html` Macros: • **`@USERNAME@`** • **`@NUM_REQUIRED_ANSWERS@`** • **`@NUM_REGISTERED_QUESTIONS@`** • **`@REQ_ADDR@`** • **`@TIMESTAMP@`**	Displayed when the user did not register the required number of knowledge questions and answers that are required for successful authentication. One of the following conditions must also be true: • The administrator did not configure the environment to allow grace-period authentication. • The user reached the limit of grace-period logins.

Authentication error template files

These files display errors that occur during authentication.

Authentication error template files

These files display errors that occur during authentication.

Table 118. Default files in the `proper/` directory

Page name	File name and macros	Description
Access Denied	`proper/errors/access_denied.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`**	Displays a message that the user cannot access the requested resource.

Table 118. Default files in the `proper/` directory (continued)

Page name	File name and macros	Description
General Error	`proper/errors/allerror.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@DETAIL@`** • **`@EXCEPTION_STACK@`**	Displays general errors that are not displayed in other template files.
Missing Component Error	`proper/errors/missingcomponent.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@DETAIL@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays an error that the component required to process the request was not correctly configured or was not initialized.
Authentication Required	`proper/errors/need_authentication.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`**	Displays an error that authentication is required to access the requested resource.
Protocol Determination Error	`proper/errors/noprotdet.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays an error that the access request is to an unknown address. The error might occur because no configured endpoint or protocol exists that is mapped to this endpoint.
Protocol Runtime Error	`proper/errors/protocol_error.html` Macros: • **`@REQ_ADDR@`** • **`@TIMESTAMP@`** • **`@EXCEPTION_MSG@`** • **`@EXCEPTION_STACK@`**	Displays errors that an error occurred fulfilling a request to a specified address, and the error was caused by an unexpected error on the protocol module.

OAuth template files

These files support OAuth.

OAuth template files

These files support OAuth. For more information, see OAuth 2.0 and OIDC Support.

Table 119. Default files in the `oauth20/` directory

Page name	File name and macros	Description
OAuth 2.0 Trusted Clients Manager	`oauth20/clients_manager.html` Macros: • **`@USERNAME@`** • **`@OAUTH_CLIENT_COMPANY_NAME@`** • **`@PERMITTED_SCOPES@`** • **`@OAUTH_CUSTOM_MACRO@`**	Used by resource owners to show and manage trusted clients information.
OAuth 2.0 - Consent to Authorize	`oauth20/user_consent.html` Macros: • **`@USERNAME@`** • **`@OAUTH_CLIENT_COMPANY_NAME@`** • **`@PERMITTED_SCOPES@`** • **`@OAUTH_CUSTOM_MACRO@`**	Used by the authorization server to determine and store user consent information about which OAuth clients are authorized to access the protected resource. The page also lists of scopes that the OAuth client requests. These lists are shown in the consent page and can be of indeterminate length. The template supports multiple copies of stanzas that are repeated once for each scope in the lists.
OAuth 2.0 - Error	`oauth20/user_error.html` Macros: • **`@OAUTH_CLIENT_COMPANY_NAME@`** • **`@CLIENT_TYPE@`** • **`@CLIENT_ID@`** • **`@REDIRECT_URI@`** • **`@STATE@`** • **`@RESPONSE_TYPE@`** • **`@USERNAME@`** • **`@OAUTH_TOKEN_SCOPE_REPEAT@`** • **`@OAUTH_OTHER_PARAM_REPEAT@`** • **`@OAUTH_OTHER_PARAM_VALUE_REPEAT@`**	Shows detailed text information when an error occurs in an OAuth 2.0 flow.

Table 119. Default files in the `oauth20/` directory (continued)

Page name	File name and macros	Description
OAuth - Response	`oauth20/user_response.html` Macros: • **`@OAUTH_CODE@`**	Displays the authorization code of an OAuth client that did not specify a client redirection URI upon registration. When the OAuth client does not specify a client redirection URI or cannot receive redirects, the authorization server does not know where to send the resource owner after authorization. As a result, the OAuth client does not receive the authorization code that is required to exchange for an access token or refresh token. The page includes several codes: • The authorization code that the resource owner can provide to the trusted OAuth client. • The authorization code as machine-readable Quick Response (QR) code. **Note:** The encoder that creates the QR code follows the ISO/IEC 18004:2006 specification. Scanners that support this specification can read the generated QR code.

Customizing SAML 2.0 pages

Verify Access generates files that are displayed in response to events that occur during single sign-on requests. The response that is displayed might be a form, such as when login information is required, or an error or information statement about a condition that occurred while the request was processed.

You can customize the event pages by modifying their appearance or content.

Before you continue with the customization, you need to have a thorough understanding of how event pages are generated and displayed.

Generation of event pages

Event pages are displayed in response to events that occur during single sign-on requests. They usually contain a form (such as a prompt for user name and password information) or text (such as an informational or error message).

Event pages are dynamic pages that are generated by Security Verify Access by using the following information:

Template files

XML or HTML files that are provided with the appliance and contain elements, such as fields, text, or graphics, and sometimes macros that are replaced with information that is specific to the request or to provide a response to the request.

Page identifiers
Event information that corresponds to one or more template files. Each page identifier corresponds to a specific event condition, such as a specific error or a condition in which a message or a form must be displayed.

Message catalogs
Text that is used to replace macros in the template files.

When a request is received, the appropriate response page is generated as follows:

1. Processing of the request occurs and a response to an event is required.
2. Template files and page identifiers are read from the file system.
3. Macros in the template files are replaced with values that are appropriate for the response that is needed.
4. An appropriate event page is generated.
5. The generated event page is displayed.

SAML 2.0 page identifiers

The SAML 2.0 runtime can display HTML pages in response to events that occur during single sign-on requests. You can select which pages to display and also modify the pages.

Use HTML pages for the following purposes:

- Displaying success and error messages to users
- Asking users for confirmation
- Sending SAML messages

You can customize these HTML pages so that they display what you want. These pages contain *macros* and are similar to other HTML pages in Security Verify Access. A macro is text in an HTML page that is replaced with context-specific information. For example, the macro `@ERROR_MESSSAGE@` is replaced by text that describes the error that occurred.

You can find the SAML 2.0 pages in the local management interface using these steps:

1. Click **Federation** > **Global Settings** > **Template Files**.
2. Expand the locale folder to locate a template file.

For example, the English version of the SAML `consent_to_federate.html` template is in `C/saml20`.

All of the available SAML 2.0 HTML pages are listed in the following table.

Table 120. SAML 2.0 HTML page identifiers and macros

Page identifier	Description	Macros and descriptions
`saml20/consent_to_federate.html`	Displays during the SAML single sign-on flow whenever the service provider wants to federate the account at the identity provider with the account at the service provider.	**`@TOKEN:form_action@`** The URL to which the SAML message is sent. **`@TOKEN:SPProviderID@`** The ID of the Service Provider. **`@TOKEN:SPDisplayName@`** The name of the Service Provider. **`@TOKEN:IPProviderID@`** The name of the Identity Provider.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/logout_partial_success.html`	Displays whenever the SAML single log out flow completes with partial success.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation.
`saml20/logout_success.html`	Displays whenever the SAML single log out flow completes successfully.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation.
`saml20/nimgmt_terminate_success.html`	Displays whenever the SAML name identifier management terminate flow completes successfully.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation. **`@TOKEN:PartnerID@`** The ID of the partner.
`saml20/nimgmt_update_success.html`	Displays whenever the SAML name identifier management update flow completes successfully.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@TOKEN:UserName@`** The user name that performs the operation. **`@TOKEN:PartnerID@`** The ID of the partner.
`saml20/saml_post_artifact.html`	Sends the SAML artifact to the partner for HTTP POST binding.	**`@TOKEN:form_action@`** The URL to which the SAML message is sent. **`@TOKEN:RelayState@`** The RelayState. **`@TOKEN:SamlMessage@`** The SAML message.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/saml_post_request.html`	Sends the SAML request message to partner for HTTP POST binding.	**`@TOKEN:form_action@`** The URL to which the SAML message is sent. **`@TOKEN:RelayState@`** The RelayState. **`@TOKEN:SamlMessage@`** The SAML message.
`saml20/saml_post_response.html`	Sends the SAML response message to the partner for HTTP POST binding.	**`@TOKEN:form_action@`** The URL to which the SAML message is sent. **`@TOKEN:RelayState@`** The RelayState. **`@TOKEN:SamlMessage@`** The SAML message.
`saml20/art_exchange_failed.html`	Displays whenever there is a failure during the SAML artifact resolution flow.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.
`saml20/authn_failed.html`	Displays whenever there is a failure during the SAML single sign-on flow.	**`@REQ_ADDR@`** The URL of the request. **`@TIMESTAMP@`** The time stamp of the request. **`@ERROR_MESSAGE@`** The error message. **`@EXCEPTION_STACK@`** The stack trace of the error. Do not use this macro in a production environment.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/error_building_msg.html`	Displays whenever an outgoing SAML message is not constructed.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_decrypting_msg.html`	Displays whenever an incoming SAML message is decrypted.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_missing_config_param.html`	Displays whenever a SAML flow is run on a SAML federation with invalid configuration.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/error_parsing_art.html`	Displays whenever an incoming SAML artifact is parsed.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/ error_parsing_msg.html`	Displays whenever an incoming SAML message is parsed.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/ error_sending_msg.html`	Displays whenever an outgoing SAML message is sent.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/ error_validating_art.html`	Displays whenever an incoming SAML artifact is validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/ error_validating_init_msg.html`	Displays whenever a SAML flow is initiated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions

`saml20/ error_validating_msg.html`	Displays whenever an incoming SAML message is validated.	**`@REQ_ADDR@`**
		The URL of the request.
		`@TIMESTAMP@`
		The time stamp of the request.
		`@ERROR_MESSAGE@`
		The error message.
		`@EXCEPTION_STACK@`
		The stack trace of the error. Do not use this macro in a production environment.

`saml20/ error_validating_msg_signature.html`	Displays whenever an incoming SAML message is signature validated.	**`@REQ_ADDR@`**
		The URL of the request.
		`@TIMESTAMP@`
		The time stamp of the request.
		`@ERROR_MESSAGE@`
		The error message.
		`@EXCEPTION_STACK@`
		The stack trace of the error. Do not use this macro in a production environment.

`saml20/invalid_art.html`	Displays whenever an incoming SAML artifact is validated.	**`@REQ_ADDR@`**
		The URL of the request.
		`@TIMESTAMP@`
		The time stamp of the request.
		`@ERROR_MESSAGE@`
		The error message.
		`@EXCEPTION_STACK@`
		The stack trace of the error. Do not use this macro in a production environment.

`saml20/ invalid_init_msg.html`	Displays whenever a SAML flow is initiated.	**`@REQ_ADDR@`**
		The URL of the request.
		`@TIMESTAMP@`
		The time stamp of the request.
		`@ERROR_MESSAGE@`
		The error message.
		`@EXCEPTION_STACK@`
		The stack trace of the error. Do not use this macro in a production environment.

Table 120. SAML 2.0 HTML page identifiers and macros (continued)

Page identifier	Description	Macros and descriptions
`saml20/invalid_msg.html`	Displays whenever an incoming SAML message is validated.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/logout_failed.html`	Displays whenever there is a failure during SAML single logout flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/nimgmt_terminate_failed.html`	Displays whenever there is a failure during the SAML name identifier terminate management flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.
`saml20/nimgmt_update_failed.html`	Displays whenever there is a failure during the SAML name identifier update management flow.	**@REQ_ADDR@** The URL of the request. **@TIMESTAMP@** The time stamp of the request. **@ERROR_MESSAGE@** The error message. **@EXCEPTION_STACK@** The stack trace of the error. Do not use this macro in a production environment.

Template page for the WAYF page

The Where Are You From (WAYF) page is used at the service provider. The WAYF page enables users to select their identity provider if there is more than one configured in the federation.

When a user arrives at a service provider, a WAYF identifier can be delivered through a cookie or query-string parameter with the request. The entity ID of the identity provider is stored as the value of the cookie or query-string parameter. If the WAYF identifier cookie or query-string parameter is not present, the WAYF page opens.

An example URL that includes the query string parameter for WAYF:

```
https://sp.host.com/isam/sps/samlfed/saml20/
logininitial?RequestBinding=HTTPRedirect&ResponseBinding
=HTTPPost&ITFIM_WAYF_IDP=https://idp.host.com/isam/sps/samlfed/saml20
```

This example is for a SAML 2.0 single sign-on URL. The query string parameter name is `ITFIM_WAYF_IDP`. The value of the identity provider ID is `https://idp.host.com/isam/sps/samlfed/saml20`.

The WAYF page requires the user to indicate where they came from. If the user is not logged on to their identity provider, they are asked to log on. Depending on the attributes passed, the service provider can grant or deny access to the service.

You can find the template pages for WAYF in the local management interface using these steps:

1. Click **Federation** > **Global Settings** > **Template Files**.
2. Expand the locale folder and navigate to `/pages/itfim/wayf`.

Administrators can use the WAYF page without modifications, but in some cases might want to modify the HTML style to match the specific deployment environment.

This template file provides several replacement macros:

@WAYF_FORM_ACTION@
This macro is replaced with the endpoint of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_METHOD@
This macro is replaced with the HTTP method of the original request. This macro does not belong within a repeatable section.

@WAYF_FORM_PARAM_ID@
This macro is replaced with ID used by the action for the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_ID@
This macro is replaced with the unique ID of the identity provider. This macro is repeated once for each identity provider.

@WAYF_IP_DISPLAY_NAME@
This macro is replaced with the configured display name of the identity provider. This macro is repeated once for each identity provider.

@WAYF_HIDDEN_NAME@
This macro is replaced with the name of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

@WAYF_HIDDEN_VALUE@
This macro is replaced with the value of the hidden parameter. This macro is repeated once for each original request parameter and is hidden.

Customizing the Consent to Federate Page

A *consent to federate page* is an HTML form which prompts a user to give consent to joining a federation. You can customize the *consent to federate page* to specify what information it requests from a user.

Before you begin

Determine what values you want to use for the consent to federate page.

About this task

When a user accesses a federation, they agree to join the federation. The HTML form `saml20/consent_to_federate.html` prompts for this consent. You can customize what the form requests by adding consent values. These values indicate how a user agrees to join a federation and if service providers are notified of the consent. Identity providers receive the consent values in the SAML 2.0 response.

The following values determine how a user joins a federation:

1
: A user agrees to join a federation without notifying the service provider.

0
: A user refuses to join a federation.

A URI value
: A URI can indicate whether the user agrees to join a federation and if you want to notify the service provider about the user consent. The following table lists and describes the supported URI values.

Table 121. Supported consent values for SAML 2.0 response

Consent value	**URI**	**Description**
Unspecified	`urn:oasis:names:tc:SAML:2.0:consent:unspecified`	The consent of the user is not specified.
Obtained	`urn:oasis:names:tc:SAML:2.0:consent: obtained`	Specifies that user consent is acquired by the issuer of the message.
Prior	`urn:oasis:names:tc:SAML:2.0:consent: prior`	Specifies that user consent is acquired by the issuer of the message before the action which initiated the message.
Implicit	`urn:oasis:names:tc:SAML:2.0:consent: current-implicit`	Specifies that user consent is implicitly acquired by the issuer of the message when the message was initiated.
Explicit	`urn:oasis:names:tc:SAML:2.0:consent: current-explicit`	Specifies that the user consent is explicitly acquired by the issuer of the message at the instance that the message was sent.
Unavailable	`urn:oasis:names:tc:SAML:2.0:consent:unavailable`	Specifies that the issuer of the message was not able to get consent from the user.
Inapplicable	`urn:oasis:names:tc:SAML:2.0:consent:inapplicable`	Specifies that the issuer of the message does not need to get or report the user consent.

Follow the steps in this procedure to customize the consent to federate page.

Procedure

1. Log in to the local management interface.
2. Click **Federation** > **Global Settings** > **Template Files**.
3. Expand a locale and select `saml20/consent_to_federate.html`.
4. Click **Edit** and add the appropriate consent values for your federation.
5. Click **Save**.
6. Deploy the changes.

Example

The following example shows an added URI with a consent value `Obtained`:

```
<input type="radio" checked name="Consent"
value="urn:urn:oasis:names:tc:SAML:2.0:consent:obtained"/>
Consent Obtained.<br/>
```

In this example, the user consent is acquired by the issuer of the message.

Template file macros

Most template pages contain one or more macros. The macros are replaced by values that are specific to the action that is requested on the page.

Macro	**Value that replaces the macro**
`@CLIENT_ID@`	The `client_id` parameter that is specified in the authorization request.
`@CONSENT_FORM_VERIFIER@`	A unique identifier for the consent_form_verifier parameter value. The value is automatically generated by the authorization server. Do not modify the parameter name or value.
`@DETAIL@`	The error message.
`@ERROR_CODE@`	Characters that uniquely identify the error.
`@ERROR_DESCRIPTION@`	The native language support (NLS) text of the error message that is associated with the error.
`@ERROR_MESSAGE@`	An error message that is specific to the action in the page. For example, on the One-time password template page for login, the error message indicates that the password submitted contains errors, such as the password is not valid or has expired.
`@EXCEPTION_MSG@`	The exception message.
`@EXCEPTION_STACK@`	The stack trace of the error.
`@GRACE_PERIOD_AUTH_COUNT@`	The amount of grace-period authentication.
`@LICENSE@`	The contents of the license file.
`@LICENSE_FILE@`	The name of the license file.
`@LICENSE_METADATA@`	The metadata that is either: • Defined in the license file.

Macro	Value that replaces the macro
	• `Not Available` if it is not defined.
`@MAPPING_RULE_DATA@`	If the submitted one-time password contains an error, this value is the STS Universal User context attribute with the name `@MAPPING_RULE_DATA@` and is type `otp.sts.macro.type`. This context attribute can be set in the OTPVerify mapping rule.
`@MAX_GRACE_PERIOD_AUTH_COUNT@`	The maximum count of grace-period authentication that is allotted to a policy.
`@MAX_STORED_QUESTIONS@`	The maximum number of answers that can be stored per user.
`@NUM_REQUIRED_ANSWERS@`	The number of valid answers that is required for successful authentication.
`@NUM_REGISTERED_QUESTIONS@`	The number of questions that the user registered.
`@OAUTH_AUTHORIZE_URI@`	The URI for the authorization endpoint.
`@OAUTH_CLIENT_COMPANY_NAME@`	A multi-valued macro that belongs inside an `[RPT trustedClients]` repeatable replacement list. The values are replaced with the name of the company that requests access to the protected resource.
`@OAUTH_CLIENTMANAGERURL@`	A multi-valued macro that belongs inside an `[RPT trustedClients]` repeatable replacement list. The values are replaced with the endpoint of the trusted clients manager.
`@OAUTH_CODE@`	The `oauth_code` parameter that is specified in the authorization response.
`@OAUTH_CUSTOM_MACRO@`	A multi-valued macro that belongs inside an `[RPT trustedClients]` repeatable replacement list. The values are replaced with trusted client information that contains additional information about an authorized OAuth client.
`@OAUTH_OTHER_PARAM_REPEAT@`	A multi-valued macro that belongs inside an `[RPT oauthOtherParamsRepeatable]` repeatable replacement list. The values show the list of extra parameter names.
`@OAUTH_OTHER_PARAM_VALUE_REPEAT@`	A multi-valued macro that belongs inside an `[RPT oauthOtherParamsRepeatable]` repeatable replacement list. The values show the list of extra parameter values.
`@OAUTH_TOKEN_SCOPE_REPEAT@`	A multi-valued macro that belongs inside an `[RPT oauthTokenScopePreapprovedRepeatable]` or `[RPT oauthTokenScopeNewApprovalRepeatable]`repeatable replacement lists. The values inside the `[RPT oauthTokenScopePreapprovedRepeatable]` show the list of token scopes that have been previously approved by the resource owner. Alternatively, the values inside the `[RPT oauthTokenScopeNewApprovalRepeatable]`

Macro	Value that replaces the macro
	show the list of token scopes that have not yet been approved by the resource owner.
@OTP_HINT@	The one-time password hint. The hint is a sequence of characters that is associated with the one-time password.
@OTP_METHOD_CHECKED@	For the first method, this macro is replaced with an HTML radio button attribute that causes that radio button to be selected. For the remaining methods that generate, deliver, and verify one-time passwords, this macro is replaced with an empty string.
@OTP_METHOD_ID@	The ID of the method for generating, delivering, and verifying the one-time password. This ID is generated by the `OTPGetMethods` mapping rule.
@OTP_METHOD_LABEL@	The label of the method for generating, delivering, and verifying the one-time password. This label is generated by the `OTPGetMethods` mapping rule.
@OTP_METHOD_TYPE@	The type of the currently selected method for generating, delivering, and verifying the one-time password. This type is generated by the `OTPGetMethods` mapping rule and was selected by the user.
@OTP_STRING@	The one-time password that is generated by the one-time password provider.
@PERMITTED_SCOPES@	A multi-valued macro that belongs inside an `[RPT trustedClients]` repeatable list. The values are replaced with the token scopes to which the OAuth client has access.
@QUESTION_COUNT@	The number of questions that are presented on the login page.
@QUESTION_TEXT@	The question text. This macro is only populated when the question is a user-provided question.
@QUESTION_INDEX@	The question index. This index corresponds to the array of questions that are presented on the page when questions are presented as a group.
@QUESTION_UNIQUE_ID@	The question unique identifier.
@REDIRECT_URI@	The redirect URI that the authorization server uses to send the authorization code to. The value depends on the following items: • Redirect URI that is entered during partner registration. • `oauth_redirect` parameter that is specified in the authorization request
@REGENERATE_ACTION@	The URl where the Generate button posts the form to regenerate and deliver the new one-time password value.

Macro	Value that replaces the macro
@RESPONSE_TYPE@	The `response_type` parameter specified in the authorization request.
@REQ_ADDR@	The URL into which the request from the user is sent.
@RESELECT_ACTION@	The URl where the Reselect button posts the form to reselect the method for generating, delivering, and verifying the one-time password value.
@STATE@	The `state` parameter that is specified in the authorization request.
@TIMESTAMP@	The time stamp when the error occurred.
@UNIQUE_ID@	A multi-valued macro that belongs inside an `[RPT trustedClients]` repeatable replacement list. The values are replaced with a unique identifier that identifies the trusted client information for each entry in the list.
@USERNAME@	The Security Verify Access user name.

Mapping rules

Mapping rules are JavaScript code that runs during the authentication flow for Advanced Access Control and Federation.

Mapping rules can be used for multiple purposes. For Advanced Access Control, you can modify rules for the Authentication Service, OTP, and OAuth 2.0. For Federation, you can modify mapping rules to manage identities for OIDC and SAML 2.0. Use the task topic below that applies to the type of mapping rule you want to manage.

Note: Support for the importing of a mapping rule into another mapping rule applies to all mapping rules.

Managing JavaScript mapping rules

Create or edit JavaScript mapping rules.

About this task

When you activate the Advanced Access Control offering, the following mapping rule types are available:

AuthSvc
Authorization service mapping rule.

OAUTH
OAuth mapping rule.

OTP
One-time password mapping rule.

OIDC
OpenID Connect mapping rule.

SAML2_0
SAML 2.0 mapping rule.

Procedure

1. Click **AAC**.

2. Under **Global Settings**, click **Mapping Rules**.
   All existing mapping rules are displayed.
3. You can create or modify a mapping rule.
   - To create a mapping rule
     a. Click **Add**.
     b. In the **Content** field, enter the JavaScript mapping rule content.
     c. In the **Name** field, enter a name for the rule.
     d. In the **Category** field, select the type of the mapping rule from the list.

        Note: Only the mapping rule types that apply to your current activated offering are displayed in the list.
     e. Click **Save**.
   - To modify a mapping rule
     a. Select the mapping rule to modify.
     b. Click **Edit**.
     c. Modify the mapping rule in the **Content** field as needed.

        Note: The **Name** and **Category** fields are not editable.
     d. Click **Save**.

Authentication Service Credential mapping rule

The Authentication Service Credential mapping rule is JavaScript code that you can use to customize the information that is contained in the user credential.

During authentication, the Authentication Service gathers information about the authenticated user, including attributes associated with the user ID. After successful authentication, the Authentication Service provides this information to the Authentication Service Credential mapping rule. The main task of the mapping rule is to modify or add attributes to the user information before it is used to generate a credential.

Customizing the mapping rule is an advanced way to customize the credential. To specify basic credential attributes, use an authentication policy and the **Credentials** panel in the local management interface instead of creating a custom mapping rule. See Creating an authentication policy.

If you write your own mapping rule and use it to replace the existing rule, be aware of the following considerations:

- Credential attributes are string values. For example, user names and lists of groups are string arrays.
- Do not use spaces, commas, or colons in credential attribute names. Use alphanumeric characters.

The sample mapping rule provides more descriptions about considerations for writing your own mapping rule.

A default `AuthSvcCredential` mapping rule is provided. To review the rule:

1. Log in to the local management interface.
2. Click **AAC**
3. Under **Policy**, click **Authentication**.
4. Click **Advanced**.
5. Select `AuthSvcCredential`.
6. Click 📥.
7. Choose a location and save the file.

To review an example of a customized credential mapping rule:

1. Log in to the local management interface.
2. Click **System**.
3. Click **File Downloads**.
4. Click **access_control** > **examples** > **mapping_rules**.
5. Select `authsvc_credential.js`.
6. Click **Export** to download the file.

If you create your own rule, use it to replace the existing rule. See the replacement instructions in Managing mapping rules.

OTPGetMethods mapping rule

`OTPGetMethods` specifies the methods for delivering the one-time password to the user.

This sample mapping rule sets password delivery conditions for the following delivery methods:

- By email
- By SMS
- No delivery

Each delivery method includes the following attributes and their corresponding value:

`id`
Specifies a unique delivery method ID. This value replaces the `@OTP_METHOD_ID@` macro in the **OTP Method Selection** page. Use a unique value across different methods. For example, `sms`.

`deliveryType`
Specifies the delivery plug-in that delivers the one-time password. The value must match one of the types in the `DeliveryTypesToOTPDeliveryModuleIds` parameter of the OTP response file. For example, `sms_delivery`.

`deliveryAttribute`
Specifies an attribute that is associated with the delivery type. The value depends on the one-time password provider plug-in for the delivery type. For example:

- For SMS delivery, the value is the mobile number of the user. For example, `mobileNumber`.
- For email delivery, the value is the email address of the user. For example, `emailAddress`.
- For no delivery, the value is an empty string.

`label`
Specifies the unique delivery method to the user. For time-based and counter-based one-time password, use this attribute to specify the secret key of the user. If `label` is not specified, the time-based and counter-based one-time password code retrieves the key by invoking the user information provider plug-in. This parameter replaces the `@OTP_METHOD_LABEL@` macro in the **OTP Method Selection** page.

`otpType`
Specifies the one-time password provider plug-in that generates and verifies the password. The value must match one of the types in the `OTPTypesToOTPProviderModuleIds` parameter of the OTP response file. For example, `mac_otp`.

`userInfoType`
Specifies which user information provider plug-in to use to retrieve user information that is required to calculate the one-time password. This parameter is only required if user information is used for calculation of the one-time password.

To customize one-time password delivery, you can do one of the following actions:

- Create your own mapping rules that are based on the sample `OTPGetMethods` mapping rule.
- Modify the sample `OTPGetMethods` mapping rule.

You can also customize the mapping rule to use access control context data. For details see, Customizing one-time password mapping rules to use access control context data.

OTPGenerate mapping rule

`OTPGenerate` mapping rule specifies the generation of the one-time password for the user.

You can use the `OTPGenerate` mapping rule in the following configuration:

Modify the one-time password type of the selected method to generate the one-time password
Indicates the one-time password type to determine the one-time password Provider plug-in that generates the one-time password for the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, Customizing one-time password mapping rules to use access control context data.

OTPDeliver mapping rule

The `OTPDeliver` mapping rule specifies the delivery method of the one-time password to the user.

Use the following `OTPDeliver` mapping rules:

Generate the one-time password hint
The one-time password hint is a sequence of characters that is associated with the one-time password. The one-time password hint is displayed in the **One-Time Password Login** page. It is also sent to the user together with the one-time password.

You can customize the way the one-time password hint is generated by modifying the following section in the default `OTPDeliver` mapping rule:

```
var otpHint = Math.floor(1000 + (Math.random() * 9000));
```

Note: See the comments in the mapping rule for more details.

Generate the formatted one-time password
The formatted one-time password is the formatted version of the one-time password. The formatted one-time password, instead of the actual one-time password, is sent to the user. For example, for one-time password hint `abcd`, and one-time password `12345678`, you can set the formatted one-time password as `abcd-12345678`. For one-time password hint `efgh`, and one-time password`87654321`, you can set the one-time password as `efgh#8765#4321`.

You can customize the way that the one-time password is generated by modifying the following section in the sample OTPDeliver mapping rule:

```
var otpFormatted = otpHint + "-" + otp;
```

Note: See the comments in the mapping rule for more details.

Modify the delivery type of the selected method for delivering the one-time password
The delivery type specifies the one-time password Delivery plug-in that delivers the one-time password to the user.

Modify the delivery attribute of the selected method to deliver
The delivery attribute is an attribute that is associated with delivery type. The meaning of the delivery attribute depends on the one-time password provider plug-in for the delivery type. For example, for SMS delivery type, the delivery attribute is the mobile number of the user. For email delivery type, the delivery attribute is the email address of the user.

Note: See the comments in the mapping rule for more details.

You can also customize the mapping rule to use access control context data. For details see, Customizing one-time password mapping rules to use access control context data.

OTPVerify mapping rule

`OTPVerify` specifies the verification of the one-time password that is submitted by the user.

You can customize the sample `OTPVerify` mapping rule to modify the following verification rules:

Modify the one-time password type of the user

Indicates the one-time password type to determine the one-time Provider plug-in that verifies the one-time password submitted by the user.

Set the authentication level of the user

After one-time password authentication completes, a credential is issued that contains the authentication level of the user. You can customize the authentication level by modifying the following section in the mapping rule:

```
var authenticationLevel = contextAttributesAttributeContainer.getAttributeValueByNameAndType
       ("otp.otp-callback.authentication-level", "otp.otp-callback.type");
var attributeAuthenticationLevel = new Attribute("AUTHENTICATION_LEVEL",
       "urn:ibm:names:ITFIM:5.1:accessmanager", authenticationLevel);
attributeContainer.setAttribute(attributeAuthenticationLevel);
```

Enforce the number of times the user can submit the one-time password in the one-time password login page

If a user exceeds the permitted number of times to submit a one-time password, an error message displays. You can customize the number of times that the user can submit the one-time password in the one-time password login page by modifying the following section in the mapping rule:

```
var retryLimit = 5;
```

By default, this option is set to `false`.

Note: This setting applies only to MAC OTP.

Identify the secret key of a user

When a user registers with a time-based one-time password application, they are assigned a secret key. Store the secret key in this mapping rule for verification of the user by modifying the following code:

```
var secretStr = new java.lang.String(SECRET_KEY_GOES_HERE);
```

By default, this option is set to `false`.

Override the one-time password target URL

By default, a user is redirected to a target URL upon completion of an one-time password flow. That target URL was either the initial cached request at the WebSEAL or reverse proxy instance or was specified as part of the one-time password invocation using the **`Target`** query string parameter.

You can use the `OTPVerify` mapping rule to override this target URL by adding an attribute called **`itfim_override_targeturl_attr`**. This attribute ensures that at the completion of a successful one-time password flow, the user is redirected to the override target instead of the initial target. Example code:

```
var targetUrl = new java.lang.String("http://www.example.com/url");
var targetUrlAttr = new Attribute("itfim_override_targeturl_attr",
"urn:ibm:names:ITFIM:5.1:accessmanager", targetUrl);
attributeContainer.setAttribute(targetUrlAttr);
```

To customize one-time password verification, you can do one of the following actions:

- Create your own verification rules that are based on the sample `OTPVerify` mapping rule.
- Modify the sample `OTPVerify` mapping rule.

You can also customize the mapping rule to use access control context data. For details see, Customizing one-time password mapping rules to use access control context data.

Customizing one-time password mapping rules to use access control context data

Some authentication scenarios require that context data used in making an access control decision be available during authentication. You can configure Security Verify Access to capture the content data and make it available to the one-time password mapping rules.

About this task

You can configure Security Verify Access to perform access control policy evaluation when a resource is accessed. The access control policy evaluation can result on a permit with authentication. The required authentication is determined by the access control policy. Some scenarios require that the context data used to perform the access control decision be available during the authentication. In order to provide access to the access control context data, you can persist the context information for the predefined authentication obligations that perform one-time password authentication.

Note: The context data available is limited to the attributes referenced by the access control policy and the request attributes provided by the policy enforcement point. If the policy relies on the risk score to perform access control, the context data available also includes the risk-profile attributes.

Procedure

1. Log in to the local management interface.
2. Click **AAC** > **Global Settings** > **Advanced Configuration**.
3. Select **attributeCollection.authenticationContextAttributes**.
4. Click for the property.
5. In the text field, enter a list of comma separated attribute names to be collected during the authorization policy evaluation.
   For example, if your scenario requires the authentication level and host of the request the configuration property, enter `authenticationLevel, http:host`.

   The access control context data is provided to the one-time password mapping rules as context attributes values. The following format is used:

   ```
   <stsuuser:Attribute name="AttributeName-AttributeURI"
     type=""authn.service.context.attribute.type.AttributeDatatype">
   <stsuuser:Value>AttributeValue</stsuuser:Value>
   </stsuuser:Attribute>
   ```

   Where:

   - `name` is the attribute name and attribute identifier separated by a dash (-).
   - `type` is the attribute data type prefixed by `authn.service.context.attribute.type`.

   For example the `authenticationLevel` attribute value is added as:

   ```
   <stsuuser:Attribute name="authenticationlevel-urn-ibm:
      security:subject:authenticationlevel"
      type="authn.service.context.attribute.type.Integer">
   <stsuuser:Value>1</stsuuser:Value>
   </stsuuser:Attribute>
   ```
6. Click **OK**.
7. When you edit a property, a message indicates that there are undeployed changes. If you have finished making changes, deploy them.

   For more information, see Deploying pending changes.

8. Configure the mapping rule to use the information collected by this property as the context attribute.
   a) Click **AAC**.
   b) Under **Policy**, click **Authentication**.
   c) Click **Advanced**.
   d) Select and export the mapping rule.
   e) Use a text editor and modify the rule to access the attributes collected during the access control policy evaluation in the following format:

   ```
   var accessControlAttribute =
   contextAttributesAttributeContainer.getAttributeValueByNameAndType
   ("AttributeName-AttributeURI",
   "authn.service.context.attribute.type.AttributeDatatype");
   ```

   Where:

   - `name` is the attribute name and attribute identifier separated by a dash (-).
   - `type` is the attribute data type prefixed by `authn.service.context.attribute.type`.

   For example, the `authenticationLevel` attribute can be obtained using the following information:

   ```
   var accessControlAuthenticationLevel =
   contextAttributesAttributeContainer.getAttributeValueByNameAndType
   ("authenticationlevel-urn-ibm:security:subject:authenticationlevel",
   "authn.service.context.attribute.type.Integer");
   ```

   f) Save the mapping rule and take note of its location.
   g) In the local management interface, click **AAC**.
   h) Under **Policy**, click **Authentication**.
   i) Click **Advanced**.
   j) Select the mapping rule you want to replace.
   k) Click **Replace**. The Replace Mapping Rule panel opens.
   l) Click the field or the **Browse** button and select the file for your saved mapping rule.

   Attention: The name of the mapping rule cannot be replaced. The name of the uploaded file is ignored.

   m) Click **OK** to upload the mapping rule.

Managing OAuth 2.0 mapping rules

Use the mapping rules to customize the methods for the OAuth 2.0 or OIDC flow.

About this task

The OAuth 2.0 and OIDC mapping rules are JavaScript code that run during the OAuth 2.0 or OIDC flow. You can view, export, and replace OAuth or OIDC mapping rules.

View the mapping rule if you want to see the content and structure of the mapping rule. Export the mapping rule if you want to save a copy of the mapping rule. You can also edit this copy. Replace a mapping rule if you want to use a new mapping rule.

Procedure

1. Log in to the local management interface.
2. Click **AAC** > **Policy** > **OpenID Connect and API Protection** or **Federation** > **Manage** > **OpenID Connect and API Protection**.
3. Click **Mapping Rules**.

4. Perform one or more of the following actions:

   View a mapping rule

   a. Select a mapping rule.
   b. Click . The **View Mapping Rule** panel opens. The content of the mapping rule is displayed.
   c. Click **OK** to close the panel.

   Export a mapping rule

   a. Select a mapping rule.
   b. Click .
   c. Choose a location and save the file.

   Replace a mapping rule:

   Note: Use an existing mapping rule as the basis for the updated mapping rule.

   a. Select a mapping rule that you want to replace.
   b. Click . The **Replace Mapping Rule** panel opens.
   c. Click the field or **Browse** and select a file.
   d. Click **OK** to upload the mapping rule.
5. When you replace a mapping rule, the appliance displays a message that there are undeployed changes. If you are finished with the changes, deploy them.

   For more information, see Deploying pending changes.

Related reference
OAuth 2.0 and OIDC mapping rule methods

OAuth 2.0 mapping rule methods

You can use Java methods to customize the `PreTokenGeneration` and `PostTokenGeneration` mapping rules.

The sample mapping rules are `oauth_20_pre_mapping.js` and `oauth_20_post_mapping.js`.

You can access the sample mapping rules from the LMI. Navigate to **System** > **Secure Settings** > **File Downloads**. Continue to either of the following locations:

- **access_control** > **examples** > **mapping rules**
- **federation** > **examples** > **mapping rules**

The following limitations affect the attribute keys and values that are associated with the `state_id` by using the `OAuthMappingExtUtils` class:

- Keys cannot be null or empty.
- Values cannot be null but can be empty.
- Associated key-value pairs are read and write-allowed and not-sensitive.
- Some keys are reserved for system use and cannot be modified by this utility. For example, the keys and values for the API PIN protection.

For more information, see the Javadoc. In the LMI, navigate to **System** > **Secure Settings** > **File Downloads**. Continue to either **access_control** > **doc** or **federation** > **doc**.

See also JavaScript whitelist.

Actions to be performed in mapping rules

For certain grant types, you must perform these actions in the pre-token mapping rule.

Resource owner password credentials (ROPC) grant type flow

For the ROPC flow, the pre-token mapping rule is responsible for performing validation of the user name and password. This validation can be performed in various ways. The pre-defined rule that is included with the appliance provides the following examples:

- The java class **`PluginUtils`** can be used to validate a user name and password against a configured LDAP.

  To configure the LDAP to be used, see Configuring username and password authentication.
- Validate the user name and password through an HTTP callout. The mapping rule sends the user name and password to a web service. As the format of the messages is not fixed, various services (for example, REST, SOAP, SCIM) can be used for this purpose. Javadoc on the HTTP client and all other exposed Java classes available in mapping rules can be downloaded from the appliance **File Downloads** page under the path **access_control** > **doc** > **ISAM-javadoc.zip**.

JWT and SAML bearer grant type flow

For the JWT or SAML assertion bearer grant type flows, the pre-token mapping rule must perform the following actions:

- Validate the assertion, including but not limited to:
  - Validate the signature (if signed).
  - Decrypt the assertion (if encrypted).
  - Check the expiry and "not before" value of the assertion.
  - Ensure that the issuer is a trusted party.
- Extract the subject from the assertion and set the **`USERNAME`** field of the STSUU.

  The **`USERNAME`** field of the STSUU can be set via a call, for example:

  ```
  // username is a variable containing the subject of the assertion
  
  stsuu.addContextAttribute(new com.tivoli.am.fim.trustserver.sts.uuser.Attribute
  ("username","urn:ibm:names:ITFIM:oauth:rule:decision", username));
  ```

  The validation of the assertion can be performed in various ways:

  - HTTP callout to a web service. Use the HTTP client to perform this.
  - WS-Trust request to the Secure Token Service (STS).
    - A chain must be configured to consume the assertion and return the required information.
    - The **`STSClientHelper`** will be called to invoke the STS via HTTP. For more information about this class, see the Javadoc that is embedded in the appliance.

  Any attributes of the assertion can be extracted and associated to the OAuth grant to be used later. For more information about associating attributes, see OAuth 2.0 and OIDC mapping rule methods.
- The type of the username attribute added must be `"urn:ibm:names:ITFIM:oauth:rule:decision"` to ensure that only a value populated from the rule is used.

MMFA mapping rule methods

Customize the OAuth **`PreTokenGeneration`** and **`PostTokenGeneration`** mapping rules by using these methods.

Sample mapping rules are available from **System** > **Secure Settings** > **File Downloads** under the **access_control** > **examples** > **mapping rules** directory.

The following limitations affect the attribute keys and values that are associated with the **`state_id`** by using the **`MMFAMappingExtUtils`** class:

- Keys cannot be null or empty.
- Values can only be null or empty when specified.
- Associated key-value pairs are read-only and not case sensitive.
- The push token is read-only and case sensitive.

`registerAuthenticator`

```
public static String registerAuthenticator(
      String stateId
      )
```

This method performs the final steps of registering an authenticator. Use the following parameters:

`stateId`

The state ID of the authorization grant. This parameter cannot be null or empty.

These responses come from the runtime after registration.

- The new authenticator's ID if successful.
- Null if not successful.

`savePushToken`

```
public static boolean savePushToken(
      String stateId,
      String pushToken,
      String applicationID
      )
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

`stateId`

The state ID of the authorization grant. This parameter cannot be null or empty.

`pushToken`

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

`applicationID`

The application ID of the authenticator application. This parameter can be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

`savePushToken`

```
public static boolean savePushToken(
      String stateId,
      String pushToken
      )
```

This method saves the push token and application ID with the authorization grant state ID. Use the following parameters:

`stateId`

The state ID of the authorization grant. This parameter cannot be null or empty.

pushToken

The push token the authenticator application has received from its push notification service provider. This parameter cannot be null or empty.

These responses come from the runtime.

- True if successful.
- False if not successful.

saveDeviceAttributes

```
public static boolean saveDeviceAttributes(
      String stateId,
      String deviceName,
      String deviceType,
      String osVersion,
      String fingerprintSupport,
      String frontCameraSupport,
      String tenantId
      )
```

This method saves various device attributes with the authorization grant state ID. Use the following parameters:

stateId

The state ID of the authorization grant. This parameter cannot be null or empty.

deviceName

The name of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

deviceType

The type of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

osVersion

The OS version of the device the authenticator is installed on. This parameter can be null or empty. If empty, the value is cleared.

fingerprintSupport

The type of fingerprint sensor that is supported by the device. This parameter can be null or empty. If empty, the value is cleared.

frontCameraSupport

flag that indicates if the device has a front facing camera. This parameter can be null or empty. If empty, the value is cleared.

tenantId

The tenant ID for this registration, if the authenticator application is multi-tenant. This parameter can be null or empty. If empty, the value is cleared.

These responses come from the runtime.

- True if successful.
- False if not successful.

JavaScript whitelist

Advanced Access Control JavaScript mapping rules and Federation mapping rules call Java code from JavaScript. The set of classes that can be called is restricted.

Exercise reasonable caution when you call Java code from JavaScript rules to ensure that accidental damage to appliance resources is avoided.

Common classes allowed in one-time password, OAuth or API protection, dynamic attributes, and JavaScript PIP, federation mapping rules, and access policies.

```
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Class
java.lang.Double
java.lang.Float
java.lang.Integer
java.lang.Long
java.lang.reflect.Array
java.lang.Short
java.lang.String
java.lang.System
```

```
java.io.ByteArrayInputStream
java.io.ObjectInputStream
java.io.PrintStream
```

```
java.math.BigDecimal
```

```
java.util.ArrayList **
java.util.Base64
java.util.Base64$Decoder
java.util.Base64$Encoder
java.util.Date
java.util.HashSet **
java.util.HashMap **
java.util.Iterator
java.util.List
java.util.logging.Level
java.util.Map
java.util.Set
java.util.UUID
```

```
com.ibm.security.access.httpclient.HttpClient
com.ibm.security.access.httpclient.HttpResponse
com.ibm.security.access.httpclient.Headers
com.ibm.security.access.httpclient.Parameters
com.ibm.security.access.httpclient.HttpClientV2
com.ibm.security.access.httpclient.RequestParameters
com.ibm.security.access.scimclient.ScimClient
com.ibm.security.access.scimcleint.ScimConfig
com.ibm.security.access.ciclient.CiClient
com.tivoli.am.rba.attributes.AttributeIdentifier
com.tivoli.am.rba.extensions.RBAExtensions
com.tivoli.am.rba.fingerprinting.ValueContainerIdentifierAdapter
com.tivoli.am.rba.extensions.Attribute$Category
com.tivoli.am.rba.extensions.Attribute$DataType
com.tivoli.am.rba.extensions.Attribute
com.tivoli.am.rba.extensions.PluginUtils
```

** Inner classes for these classes are not supported. Methods that involve an inner class implementation of an interface are not available. For example, do not use the following methods in `java.util.HashMap`:

- `Collection<V> values()`
- `Set<K> keySet()`
- `Set<Map.Entry<K,V>> entrySet()`

For more information about dynamic attributes, see Dynamic attributes.

For information about federation mapping rules, see "Mapping rules" on page 254.

Additional classes allowed in one-time password, OAuth or API protection mapping rules, federation mapping rules, and access policies

```
com.tivoli.am.fim.base64.BASE64Utility
com.tivoli.am.fim.fedmgr2.trust.util.LocalSTSClient
com.tivoli.am.fim.fedmgr2.trust.util.LocalSTSClient$LocalSTSClientResult
com.tivoli.am.fim.saml20.protocol.extension.js.JSMessageExtensionContext
com.tivoli.am.fim.trustserver.sts.modules.http.stsclient.STSClientHelper
com.tivoli.am.fim.trustserver.sts.oauth20.Client
com.tivoli.am.fim.trustserver.sts.oauth20.Grant
com.tivoli.am.fim.trustserver.sts.oauth20.Token
com.tivoli.am.fim.trustserver.sts.oauth20.Definition
com.tivoli.am.fim.trustserver.sts.oauth20.OidcDefinition
com.tivoli.am.fim.trustserver.sts.STSModuleException
com.tivoli.am.fim.trustserver.sts.STSUniversalUser *
com.tivoli.am.fim.trustserver.sts.utilities.HttpResponse
com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils
com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtCacheDMAPImpl
com.tivoli.am.fim.trustserver.sts.utilities.InfoCardClaim
com.tivoli.am.fim.trustserver.sts.utilities.MMFAMappingExtUtils
com.tivoli.am.fim.trustserver.sts.utilities.OAuthMappingExtUtils
com.tivoli.am.fim.trustserver.sts.utilities.QueryServiceAttribute
com.tivoli.am.fim.trustserver.sts.utilities.USCContextAttributesHelper
com.tivoli.am.fim.trustserver.sts.uuser.Attribute *
com.tivoli.am.fim.trustserver.sts.uuser.AttributeList *
com.tivoli.am.fim.trustserver.sts.uuser.AttributeStatement *
com.tivoli.am.fim.trustserver.sts.uuser.ContextAttributes *
com.tivoli.am.fim.trustserver.sts.uuser.Group *
com.tivoli.am.fim.trustserver.sts.uuser.Principal *
com.tivoli.am.fim.trustserver.sts.uuser.RequestSecurityToken *
com.tivoli.am.fim.trustserver.sts.uuser.Subject *
com.tivoli.am.fim.utils.IteratorWrapper
com.tivoli.am.rba.pip.JavaScriptPIP
com.tivoli.am.rba.pip.JavaScriptPIP$Context
java.mail.internet.InternetAddress
```

* The white list does not contain any implementation of the interfaces that are defined in the `org.w3c.dom` package. For example, you cannot use the method `org.w3c.dom.Document toXML()` in `com.tivoli.am.fim.trustserver.sts.STSUniversalUser`.

Additional classes allowed in JavaScript PIP

```
com.tivoli.am.fim.base64.BASE64Utility
com.tivoli.am.rba.pip.JavaScriptPIP
com.tivoli.am.rba.pip.JavaScriptPIP$Context
com.tivoli.am.rba.rtss.AttributeLocatorImpl
```

For more information about policy information points, see Managing policy information points.

Additional classes allowed in mapping rules

```
packages.com.ibm.security.access.user.UserLookupHelper
packages.com.ibm.security.access.user.User
com.ibm.security.access.ldap.utils.AttributeUtil
com.ibm.security.access.ldap.utils.AttributeUtil$AttributeGetResult
com.ibm.security.access.ldap.LdapAttributeGetResult
com.ibm.security.access.ldap.LdapModifyResult
com.ibm.security.access.ldap.LdapSearchResult
com.ibm.security.access.ldap.LdapContextCreateResult
com.sun.jndi.ldap.LdapSearchEnumeration
javax.naming.NamingEnumeration
javax.naming.directory.BasicAttributes
javax.naming.directory.BasicAttribute
javax.naming.directory.SearchResult
com.ibm.security.access.recaptcha.RecaptchaClient
com.ibm.security.access.signing.SigningHelper
javax.crypto.SecretKey
javax.crypto.SecretKeyFactory
javax.crypto.spec.PBEKeySpec
com.ibm.crypto.provider.PBEKey
com.ibm.crypto.provider.PBKDF2KeyImpl
com.ibm.ws.logging.internal.impl.BaseTraceService$TeePrintStream
com.tivoli.am.fim.email.Email
com.tivoli.am.fim.email.EmailDeliveryException
com.tivoli.am.fim.email.EmailSender
com.tivoli.am.fim.email.EmailSender$SendStatus
```

For information on mapping rules, see:

- Managing OAuth 2.0 and OIDC mapping rules
- Managing mapping rules

Additional classes to manage server connections

```
com.ibm.security.access.server_connections.LdapServerConnection
com.ibm.security.access.server_connections.LdapServerConnection$LdapHost
com.ibm.security.access.server_connections.ServerConnection
com.ibm.security.access.server_connections.ServerConnectionFactory
com.ibm.security.access.server_connections.SmtpServerConnection
com.ibm.security.access.server_connections.WebServerConnection
com.ibm.security.access.server_connections.CiServerConnection
```

For more information, see Managing server connections.

Classes to use with InfoMap

```
com.tivoli.am.fim.authsvc.action.authenticator.infomap.InfoMapResult
com.tivoli.am.fim.authsvc.action.authenticator.infomap.InfoMapString
```

For more information, see Configuring an Info Map authentication mechanism.

Classes to use in Access Policies

`com.ibm.security.access.policy.Context`
`com.ibm.security.access.policy.Cookie`
`com.ibm.security.access.policy.decision.ChallengeDecisionHandler`
`com.ibm.security.access.policy.decision.DecisionHandler`
`com.ibm.security.access.policy.decision.DenyDecisionHandler`
`com.ibm.security.access.policy.decision.Decision`
`com.ibm.security.access.policy.decision.DecisionType`
`com.ibm.security.access.policy.decision.HtmlPageChallengeDecisionHandler`
`com.ibm.security.access.policy.decision.HtmlPageDecisionHandler`
`com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler`
`com.ibm.security.access.policy.decision.RedirectChallengeDecisionHandler`
`com.ibm.security.access.policy.decision.RedirectDecisionHandler`
`com.ibm.security.access.policy.decision.RedirectDenyDecisionHandler`
`com.ibm.security.access.policy.oauth20.AuthenticationContext`
`com.ibm.security.access.policy.oauth20.AuthenticationRequest`
`com.ibm.security.access.policy.oauth20.Claim`
`com.ibm.security.access.policy.oauth20.ProtocolContext`
`com.ibm.security.access.policy.ProtocolContext`
`com.ibm.security.access.policy.Request`
`com.ibm.security.access.policy.saml20.AuthnRequest`
`com.ibm.security.access.policy.saml20.ProtocolContext`
`com.ibm.security.access.policy.saml20.RequestedAuthnContext`
`com.ibm.security.access.policy.Session`
`com.ibm.security.access.policy.user.Attribute`
`com.ibm.security.access.policy.user.Group`
`com.ibm.security.access.policy.user.User`
 	
For more information, see "Access policies" on page 290.

Additional classes to customize FIDO2 flows

`com.tivoli.am.fim.fido.mediation.FIDO2Registration`
`com.tivoli.am.fim.fido.mediation.FIDO2RegistrationHelper`
`com.tivoli.am.fim.fido.server.FIDOClientManager`
`com.tivoli.am.fim.fido.server.LocalFIDOClient`
 	
For more information, see FIDO2 Mediation and FIDO Client Manager

Additional classes to manage 2FA registrations

`com.tivoli.am.fim.registrations.Mechanism`
`com.tivoli.am.fim.registrations.MechanismList`
`com.tivoli.am.fim.registrations.MechanismRegistrationHelper`
`com.tivoli.am.fim.registrations.cloud.CloudMechanism`
`com.tivoli.am.fim.registrations.local.FIDORegistration`
`com.tivoli.am.fim.registrations.local.MMFARegistration`
`com.tivoli.am.fim.registrations.local.HOTPRegistration`
`com.tivoli.am.fim.registrations.local.TOTPRegistration`
`com.tivoli.am.fim.registrations.local.KnowledgeQuestionRegistration`
`com.tivoli.am.fim.registrations.local.EULAStatus`

Related tasks

Managing OAuth 2.0 and OIDC mapping rules

Managing mapping rules

Managing JavaScript mapping rules

Create, edit, or delete JavaScript mapping rules.

About this task

When you activate the Federation offering, the following mapping rule types are available:

OIDC
OpenID Connect mapping rule.

SAML2_0
SAML 2.0 mapping rule.

Procedure

1. Click **Federation**.
2. Under **Global Settings**, click **Mapping Rules**.
   All existing mapping rules are displayed.
3. You can create, edit, or delete a mapping rule.
   - To create a mapping rule
     a. Click **Add**.
     b. In the **Content** field, enter the JavaScript mapping rule content.
     c. In the **Name** field, enter a name for the rule.
     d. In the **Category** field, select the type of the mapping rule from the list, or type a name to create your own mapping rule type.

        Note: Only the mapping rule types that apply to your current activated offering are displayed in the list.
     e. Click **Save**.
   - To modify a mapping rule
     a. Select the mapping rule to modify.
     b. Click **Edit**.
     c. Modify the mapping rule in the **Content** field as needed.

        Note: The **Name** and **Category** fields are not editable.
     d. Click **Save**.
   - To delete a mapping rule

     Note: Do not delete a mapping rule that is currently used by a SAML 2.0 or OpenID Connect federation.

     a. Select the mapping rule to delete.
     b. Click **Delete**.
     c. Confirm the delete operation.

Customizing SAML 2.0 identity mapping

Use mapping rules to map local identities to SAML tokens and to map SAML tokens to local identities.

You can use an attribute source, such as LDAP, for the identity mapping. See Managing attribute sources.

You can use an HTTP external user mapping to map a local identity to a SAML token and to map SAML token to a local identity.

See Managing JavaScript mapping rules for information about how to create or modify mapping rules.

Mapping a local identity to a SAML 2.0 token

You can map a local identity to a SAML 2.0 token for an identity provider.

The Security Verify Access server places the local user identity information into an XML document that conforms to the security token service universal user (STSUUSER) schema. The identity provider issues a SAML 2.0 token to the service provider. It generates the SAML 2.0 token based on the local identity of the user. You can customize how the local identity is converted into a SAML 2.0 token by using a mapping rule.

Security Verify Access first converts the local identity to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a SAML 2.0 token.

Your mapping rule does not operate directly on local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modification that you make to an STS Universal User has an impact on the output SAML 2.0 token.

The mapping rule is responsible for the following tasks:

1. Mapping Principal Attr Name to a Principal Name entry. When the token module generates the token, this Principal name is not directly used. Instead, the value in the **Name** field is sent as input to the alias service. The alias service obtains the alias name, name identifier, for the principal, and places the returned alias in the generated token module.

   The type must be valid for SAML. For example:

   ```
   urn:oasis:names:tc:SAML:2.0:assertion
   ```

2. Setting the authentication method to the `password` mechanism. This action is required by the SAML standard.
3. Setting the audience of the audience restriction condition to the value of the STSUU element `AudienceRestriction`. If this STSUU element is not present, the audience is set to the Provider ID of the federation partner.
4. Populating the attribute statement of the assertion with the attributes in the AttributeList in the In-STSUU. This information becomes custom information in the token.

   Custom attributes might exist that are required by applications that use information that is to be transmitted between federation partners.

5. Specifying whether the assertion conditions should contain the `<saml:OneTimeUse></saml:OneTimeUse>` element. If so, insert a special context attribute into the STSUU as shown:

```
var oneTimeUseAttr = new Attribute("AssertionIncludeOneTimeUse","urn:oasis:names:tc:SAML:2.0:assertion",
 "true");
stsuu.addContextAttribute(oneTimeUseAttr);
```

6. Setting the `NameID` attribute in the assertion with Transient NameId format. This action is useful when you want to specify a name value to use instead of the default UUID that is generated by the runtime for Transient NameID format.

   To replace the UUID, create a principal name attribute of type `urn:oasis:names:tc:SAML:2.0:nameid-format:transient`, with its value provided by user.

   The examples below show the user-provided value *`UserGeneratedTransientId`* but it could be any other value. The value of the specified STSUU principal name will be set as the `NameID` in the SAML assertion.

   Example mapping rule

   ```
   importPackage(Packages.com.tivoli.am.fim.trustserver.sts.uuser);
   var transientNameId = "UserGeneratedTransientId";
   stsuu.addPrincipalAttribute(new Attribute("name",
      "urn:oasis:names:tc:SAML:2.0:nameid-format:transient", transientNameId));
   ```

   Example STSUU values after mapping rule applied

   ```
    <stsuuser:Attribute name="name" type="urn:oasis:names:tc:SAML:2.0:nameid-format:transient">
              <stsuuser:Value>UserGeneratedTransientId</stsuuser:Value>
          </stsuuser:Attribute>
   ```

   Example SAML assertion `NameID` with Transient NameId formats

   ```
   <saml:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:transient"
                            NameQualifier="https://ip-wga/isam/sps/saml20ip/saml20"
                            SPNameQualifier="https://sp-wga/isam/sps/saml20sp/saml20"
   ```

```
                        >UserGeneratedTransientId</saml:NameID>
```

7. Determine if the partner requires a specific `SPNameQualifier` within `NameID` of assertion for transient identifiers. To change `SPNameQualifer` within `NameID` of assertion, insert a special context attribute into the STSUU with a value agreed with partner as shown in the following example:

```
var SPNameQualifierAttr = new
Attribute("AssertionChangeSPNameQualifier","urn:oasis:names:tc:SAML:2.0:assertion","http://
sp/target/app");
stsuu.addContextAttribute(SPNameQualifierAttr);
```

Mapping a SAML 2.0 token to a local identity

You can map a SAML 2.0 token to a local identity for a service provider.

A service provider consumes a SAML 2.0 token that is issued by an identity provider. It generates the local identity of the user based on a SAML 2.0 token. You can customize how a SAML 2.0 token is converted into the local identity of the user by using a mapping rule.

Security Verify Access first converts a SAML 2.0 token to an STS Universal User. It then converts this STS Universal User into another STS Universal User by using a mapping rule that you provide. After that, it converts the latter STS Universal User to a local identity of the user.

Your mapping rule does not operate directly on the local identity or SAML 2.0 token. Instead, it operates on the STS Universal User. Any modifications that you make on the STS Universal User impacts the output local identity of the user.

STSRequest and STSResponse access using a JavaScript mapping rule

By using the Default Mapping STS Module and a JavaScript mapping rule, you can perform identity mapping. The mapping rule can access STSRequest and STSResponse objects.

The following two implicit objects and the classes required by these two objects can be exposed (for example, Java DOM, XML classes, and so on):

- STSRequest which represents the WS-Trust request
- STSResponse, which represents the WS-Trust response

Use JavaScript code `stsrequest.getRequestSecurityToken().getBase()` to get the input security token from the WS-Trust request. This returns the input security token as an instance of the Java class org.w3c.dom.Element.

Use JavaScript code `stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken` (outputSecurityToken) to set the output security token in the WS-Trust response. The outputSecurityToken is the output security token represented as an instance of Java class org.w3c.dom.Element. By default, WS-Trust response contains only one output security token. To return additional output security tokens, you can use the following JavaScript code:

```
stsresponse.addRequestSecurityTokenResponse().setRequestedSecurityToken(outputSecurityToken)
```

The examples in the following topics show the mapping to and from a base64 encoded JSON string. They use the Default Mapping module with a JavaScript mapping rule. The JavaScript mapping rule accesses the STSRequest and STSResponse objects and performs the identity mapping.

Mapping a JSON Web Token to a SAML2 token example

You can map a base64 encoded JSON string to a SAML 2 token by using a JavaScript mapping rule.

About this task

The steps show an end-to-end JSON to SAML2 mapping. "STSRequest and STSResponse access using a JavaScript mapping rule" on page 167 provides a description of this support.

Procedure

1. Create a JavaScript mapping rule by using the local management interface.
   a) Select **Federation** > **Global Settings** > **Mapping Rules**.
   b) Click **Add**.
   c) In the **Content** field, copy and paste the following code:

```
    importClass(com.tivoli.am.fim.base64.BASE64Utility);
    importClass(com.tivoli.am.fim.trustserver.sts.uuser.Attribute);

    var jwtElement = stsrequest.getRequestSecurityToken().getBase();
    var jwtText    = jwtElement.getTextContent();
    var jwtString  = new java.lang.String(BASE64Utility.decode(jwtText), "UTF-8");
    var jwt        = JSON.parse(jwtString);

    for (var name in jwt) {
      if (jwt.hasOwnProperty(name)) {
        if ("sub".equals(name)) {
          stsuu.addPrincipalAttribute(new Attribute("name",
 "urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress", jwt[name]));
        } else {
          stsuu.addAttribute(new Attribute(name,
"urn:oasis:names:tc:SAML:2.0:attrname-format:basic", jwt[name]));
        }
      }
    }
```

   d) In the **Name** field, enter `jwt_saml`.
   e) In the **Category** field, select **SAML2_0**.
   f) Click **Save** and deploy the changes.
2. Assemble the Security Token Service (STS) template.
   a) Select **Federation** > **Manage** > **Security Token Service**.
   b) Click **Templates**.
   c) Click **Add** and name the template `JSON to SAML2`. Click **OK**.
   d) Select the `JSON to SAML2` template and add the Default Map Module in Map mode and a Default SAML 2.0 token in Issue mode.
   e) Save and deploy the changes.
3. Create an STS chain that references the mapping rule and template you created in the previous steps.
   a) Within the **Security Token Service** panel, select **Module Chains**.
   b) Click **Add** to create the module chain, with the following values:

Table 122. JSON to SAML2 module chain values

Tab: Field	**Value**
Overview: Name	JSON to SAML2
Overview: Description	base64 encoded JSON string to SAML2 conversion STS chain
Overview: Template	JSON to SAML2
Lookup: Request Type	Validate
Lookup: Applies to Address	`jwtappliesto`
Lookup: Issuer Address	`jwtissuer`
Properties: Default Map Module (JavaScript file containing the identity mapping rule	`jwt_saml`
Properties: Default SAML 2.0 Token (Name of the organization issuing the assertions)	isam

Table 122. JSON to SAML2 module chain values (continued)

Tab: Field	Value
Properties: Default SAML 2.0 Token (Amount of time before the issue date that an assertion is considered valid)	60
Properties: Default SAML 2.0 Token (Amount of time that the assertion is valid after being issued)	60
Properties: Default SAML 2.0 Token (List of attribute types to include)	*

Use the defaults for all of the fields that are not specified in the table.

c) Save and deploy the changes.

4. Use **curl** to test the chain.

a) Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">jwtissuer</wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">
          <wsa:Address>jwtappliesto</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
      <wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
<JWT>ewogICJlbWFpbCI6ICJqb2huLmRvZUBleGFtcGxlLmNvbSIsIAogICJmYW1pbHlfbmFtZSI6ICJkb2UiLCAK
ICAiZ2l2ZW5fbmFtZSI6ICJqb2huIiwgCiAgImlzcyI6ICJpc2FtIiwgCiAgInN1YiI6ICIwMTIzNDU2Nzg5Igp9</
JWT>
      </wst:Base>
    </ns1:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bold embedded element, **<JWT> </JWT>**, is the input to the chain. This is a Base64 encoded JSON string that contains the following data::

```
{
 "email": "john.doe@example.com",
  "family_name": "doe",
  "given_name": "john",
  "iss": "isam",
  "sub": "0123456789"
}
```

b) Save this file as `jwt.xml`.

c) Run the following **curl** command, where `jwt.xml` is the WS-Trust 1.2 message:

```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary
@jwt.xml https://ip-rte/TrustServer/SecurityTokenService
```

The following results are returned:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
    <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-
ENV:Header>
    <soap:Body>
        <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/
ws-sx/ws-trust/200512">
            <wst:RequestSecurityTokenResponse xmlns:wsu="http://docs.oasis-open.org/
wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
            wsu:Id="uuidc1288a62-0153-1f8b-bf2a-b4c46f51cd03">
                <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                    <wsa:EndpointReference>
                        <wsa:Address>jwtappliesto</wsa:Address>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <wst:Lifetime xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
                    <wsu:Created>2016-03-29T06:56:13Z</wsu:Created>
                    <wsu:Expires>2016-03-29T06:57:13Z</wsu:Expires>
                </wst:Lifetime>
               <wst:RequestedSecurityToken>
                    <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ID="Assertion-
uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03"
                    IssueInstant="2016-03-29T06:56:13Z" Version="2.0">
                        <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-
format:entity">isam</saml:Issuer>
                        <saml:Subject>
                            <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:emailAddress">
                            0123456789</saml:NameID>
                            <saml:SubjectConfirmation
 Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                                <saml:SubjectConfirmationData
 NotOnOrAfter="2016-03-29T06:57:13Z"></saml:SubjectConfirmationData>
                            </saml:SubjectConfirmation>
                        </saml:Subject>
                        <saml:Conditions NotBefore="2016-03-29T06:55:13Z"
 NotOnOrAfter="2016-03-29T06:57:13Z">
                            <saml:AudienceRestriction>
                                <saml:Audience>jwtappliesto</saml:Audience>
                            </saml:AudienceRestriction>
                        </saml:Conditions>
                        <saml:AuthnStatement AuthnInstant="2016-03-29T06:56:13Z">
                            <saml:AuthnContext>

 <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password
                                </saml:AuthnContextClassRef>
                            </saml:AuthnContext>
                        </saml:AuthnStatement>
                        <saml:AttributeStatement>
                            <saml:Attribute Name="given_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">john</
saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="email"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue
 xsi:type="xs:string">john.doe@example.com</saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="iss"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">isam</
saml:AttributeValue>
                            </saml:Attribute>
                            <saml:Attribute Name="family_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
                                <saml:AttributeValue xsi:type="xs:string">doe</
saml:AttributeValue>
                            </saml:Attribute>
                        </saml:AttributeStatement>
                    </saml:Assertion>
                </wst:RequestedSecurityToken>
```

The JSON string is mapped into the SAML assertion, as shown by the previous bold text. The attributes in the SAML2 assertion are mapped from JSON attributes.

```
<wst:RequestedAttachedReference xmlns:wss="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd">
```

```
                    <wss:SecurityTokenReference xmlns:wss11="http://docs.oasis-open.org/
wss/oasis-wss-wssecurity-secext-1.1.xsd"
                    wss11:TokenType="http://docs.oasis-open.org/wss/oasis-wss-saml-token-
profile-1.1#SAMLV2.0">
                        <wss:KeyIdentifier
 xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
                        xmlns:wss="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">
                        Assertion-uuidc1288ae8-0153-10bd-b7ef-b4c46f51cd03</
wss:KeyIdentifier>
                    </wss:SecurityTokenReference>
                </wst:RequestedAttachedReference>
                <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</
wst:RequestType>
                <wst:Status>
                    <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/
valid</wst:Code>
                </wst:Status>
            </wst:RequestSecurityTokenResponse>
        </wst:RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

Related tasks

"Mapping a SAML2 token to a base64 encoded JSON string example" on page 172
You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

Mapping a SAML2 token to a JSON Web Token example

You can map a SAML 2 token to a base64 encoded JSON string by using a JavaScript mapping rule.

About this task

The steps show an end-to-end SAML to JSON mapping. "STSRequest and STSResponse access using a JavaScript mapping rule" on page 167 provides a description of this support.

Procedure

1. Create a JavaScript mapping rule using the local management interface.
   a) Select **Federation** > **Global Settings** > **Mapping Rules**.
   b) Click **Add**.
   c) In the **Content** field, copy and paste the following code:

```
    importClass(com.tivoli.am.fim.base64.BASE64Utility);
    importClass(com.tivoli.am.fim.trustserver.sts.utilities.IDMappingExtUtils)

    var jwt = {};

    var it = stsuu.getPrincipalAttributes();
    var jt = stsuu.getAttributes();

    while (it.hasNext()) {
      var attribute = it.next();
      var name      = new String(attribute.getName());
      var value     = new String(attribute.getValues()[0]);

      if ("name".equals(name)) {
        jwt["sub"] = value;
      } else {
        jwt[name] = value;
      }
    }

    while (jt.hasNext()) {
      var attribute = jt.next();
      var name      = new String(attribute.getName());
      var value     = new String(attribute.getValues()[0]);

      jwt[name] = value;
    }
    var document   = IDMappingExtUtils.newXMLDocument();
    var jwtString  = JSON.stringify(jwt);
    var jwtText    = document.createTextNode(BASE64Utility.encode((new 
 java.lang.String(jwtString)).getBytes("UTF-8")));
    var jwtElement = document.createElement("JWT");

    jwtElement.appendChild(jwtText);

    stsresponse.getRequestSecurityTokenResponse().setRequestedSecurityToken(jwtElement);
```

d) In the **Name** field, enter `saml_jwt`.

e) In the **Category** field, select SAML2_0.

f) Click **Save** and deploy the changes.

2. Assemble the Security Token Service (STS) template.

   a) Select **Federation** > **Manage** > **Security Token Service**.

   b) Click **Templates**.

   c) Click **Add** and name the template `SAML2 to JSON`. Click **OK**.

   d) Select the `SAML2 to JSON` template and add the Default SAML 2.0 Token in Validate mode and a Default Map Module in Map mode.

   e) Save and deploy the changes.

3. Create an STS chain that references the mapping rule and template you created in the previous steps.

   a) Within the **Security Token Service** panel, select **Module Chains**.

   b) Click **Add** to create a module chain, with the following values:

Table 123. SAML2 to JSON module chain values

Tab: Field	**Value**
Overview: Name	SAML2 to JSON
Overview: Description	SAML2 to base64 encoded JSON string conversion STS chain
Overview: Template	SAML2 to JSON
Lookup: Request Type	Validate
Lookup: Applies to Address	SAML2_AppliesTo
Lookup: Issuer Address	SAML2_Issuer
Properties: Default Map Module (JavaScript file containing the identity mapping rule	`saml_jwt`

   Use the defaults for all of the fields not in the table.

   c) Save and deploy the changes.

4. Use **`curl`** to test the chain.

   a) Send the following WS-Trust 1.2 message:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema" 
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header/>
  <SOAP-ENV:Body>
    <ns1:RequestSecurityToken xmlns:ns1="http://docs.oasis-open.org/ws-sx/ws-
trust/200512">
      <wst:RequestType xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
      http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</wst:RequestType>
      <wst:Issuer xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <wsa:Address xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">SAML2_Issuer</wsa:Address>
      </wst:Issuer>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
addressing">
          <wsa:Address>SAML2_AppliesTo</wsa:Address>
        </wsa:EndpointReference>
      </wsp:AppliesTo>
<wst:Base xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
        <saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
        ID="Assertion-uuidbcb46a39-0153-1337-8efa-fec506fb7461"
 IssueInstant="2016-03-28T10:10:53Z" Version="2.0">
          <saml:Issuer Format="urn:oasis:names:tc:SAML:2.0:nameid-format:entity">isam</
saml:Issuer>
          <saml:Subject>
            <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:emailAddress">0123456789</saml:NameID>
            <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
              <saml:SubjectConfirmationData NotOnOrAfter="2016-03-28T10:11:53Z"/>
            </saml:SubjectConfirmation>
          </saml:Subject>
          <saml:Conditions NotBefore="2016-03-28T10:09:53Z"
 NotOnOrAfter="2016-03-29T10:11:53Z">
            <saml:AudienceRestriction>
              <saml:Audience>jwt_saml</saml:Audience>
            </saml:AudienceRestriction>
          </saml:Conditions>
          <saml:AuthnStatement AuthnInstant="2016-03-28T10:10:53Z">
            <saml:AuthnContext>
              <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</
saml:AuthnContextClassRef>
            </saml:AuthnContext>
          </saml:AuthnStatement>
          <saml:AttributeStatement>
            <saml:Attribute Name="given_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
              <saml:AttributeValue xsi:type="xs:string">john</saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="email" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
              <saml:AttributeValue xsi:type="xs:string">john.doe@example.com</
saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="iss" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:basic">
              <saml:AttributeValue xsi:type="xs:string">isam</saml:AttributeValue>
            </saml:Attribute>
            <saml:Attribute Name="family_name"
 NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
              <saml:AttributeValue xsi:type="xs:string">doe</saml:AttributeValue>
            </saml:Attribute>
          </saml:AttributeStatement>
        </saml:Assertion>
      </wst:Base>
    </ns1:RequestSecurityToken>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The bold element in the SAML2 assertion is mapped to the JSON attributes in the result.

b) Save this file as `saml2.xml`.

c) Run the following **`curl`** command, where `saml2.xml` is the WS-Trust 1.2 message:

```
curl -k -v -u "easuser:passw0rd" -H "Content-Type: text/xml" --data-binary
@saml2.xml https://ip-rte/TrustServer/SecurityTokenService
```

The following results are returned:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <SOAP-ENV:Header xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"></SOAP-
ENV:Header>
    <soap:Body>
        <wst:RequestSecurityTokenResponseCollection xmlns:wst="http://docs.oasis-open.org/
ws-sx/ws-trust/200512">
            <wst:RequestSecurityTokenResponse
            xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd"
            wsu:Id="uuidc1676e30-0153-16a8-86b5-c34fd1aca7a8">
```

```
                <wsp:AppliesTo xmlns:wsa="http://www.w3.org/2005/08/addressing"
                xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                    <wsa:EndpointReference>
                        <wsa:Address>SAML2_AppliesTo</wsa:Address>
                    </wsa:EndpointReference>
                </wsp:AppliesTo>
                <wst:RequestedSecurityToken>

  <JWT>eyJzdWIiOiIwMTIzNDU2Nzg5IiwiZ2l2ZW5fbmFtZSI6ImpvaG4iLCJOb3RPbk9yQWZ0ZXIiOiIyMDE2LTAz

 LTI5VDEwOjExOjUzWiIsIkF1dGhlbnRpY2F0aW9uTWV0aG9kIjoidXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6MS4w

 OmFtOnBhc3N3b3JkIiwiZW1haWwiOiJqb2huLmRvZUBleGFtcGxlLmNvbSIsIkF1ZGllbmNlUmVzdHJpY3Rpb25

 Db25kaXRpb24uQXVkaWVuY2UiOiJqd3Rfc2FtbCIsImlzcyI6ImlzYW0iLCJJc3N1ZUluc3RhbnQiOiIyMDE2LT

 AzLTI4VDEwOjEwOjUzWiIsImZhbWlseV9uYW1lIjoiZG9lIiwiTm90QmVmb3JlIjoiMjAxNi0wMy0yOFQxMDowO

 To1M1oiLCJBdXRoZW50aWNhdGlvbkluc3RhbnQiOiIyMDE2LTAzLTI4VDEwOjEwOjUzWiIsImlzc3VlciI6Iml
                zYW0ifQ==</JWT>
                </wst:RequestedSecurityToken>
                <wst:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Validate</
wst:RequestType>
                <wst:Status>
                    <wst:Code>http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/
valid</wst:Code>
                </wst:Status>
            </wst:RequestSecurityTokenResponse>
        </wst:RequestSecurityTokenResponseCollection>
    </soap:Body>
</soap:Envelope>
```

The bold embedded element, **<JWT> </JWT>)** , is the result in a Base64 encoded JSON Web Token:

```
{
  "sub": "0123456789",
  "given_name": "john",
  "NotOnOrAfter": "2016-03-29T10:11:53Z",
  "AuthenticationMethod": "urn:oasis:names:tc:SAML:1.0:am:password",
  "email": "john.doe@example.com",
  "AudienceRestrictionCondition.Audience": "jwt_saml",
  "iss": "isam",
  "IssueInstant": "2016-03-28T10:10:53Z",
  "family_name": "doe",
  "NotBefore": "2016-03-28T10:09:53Z",
  "AuthenticationInstant": "2016-03-28T10:10:53Z",
  "issuer": "isam“
}
```

Related tasks

“Mapping a base64 encoded JSON string to a SAML2 token example” on page 168
You can map a base64 encoded JSON string to a SAML 2 token by using a JavaScript mapping rule.

OpenID Connect mapping rules

Mapping rules allow users to customize the information that is propagated from an OpenID Connect Provider or what is consumed by a Relying Party.

These mapping rules can either be JavaScript, which is invoked internally via the STS, or the mapping can be performed externally via a HTTP request.

OpenID Connect Provider mapping rules

When you write mapping rules for a provider, the primary goal is to augment the claims that are included in the ID token.

After mapping rule execution, all attributes in the STSUU will be added to the id_token as a claim, where the attribute key is the key in the id_token, and the value is the value of the attribute. If there are several attributes with the same key, then an array containing each attribute will be added to the claim. Some context information is made available to the user when writing mapping rules; the context attributes of the passed in STSUU will contain attributes with the type "urn:ibm:ITFIM:oidc:provider:context", which can be used to make decisions on what claims are added, or if any other actions are performed.

These context attributes include:

- The client ID of the client making the request.
- The federation name of the provider servicing the request.
- The redirect URI sent in the request.
- The response type of the request.
- The state parameter of the request.
- The user-consented scopes for the request.

OpenID Connect Relying Party mapping rules

When you write mapping rules for a Relying Party, the resulting STSUU is turned into a PAC that is used to authenticate the user to a Reverse Proxy via EAI.

The attributes that are included in that PAC will be the attributes of the STSUU, and the principal will be the first principal which was in the STSUU. When writing mapping rules for a Relying Party, the values of the `id_token` will be made available as Attributes in the STSUU. Some additional context is made available to the user via the STSUU's context attributes. These attributes will have the types "urn:ibm:ITFIM:oidc:client:idtoken:param" and "urn:ibm:ITFIM:oidc:client:token:param".

These context attributes include:

- All of the claims inside the `id_token`.
- The raw JWT.
- Any issued access or refresh tokens.
- All of the properties of the issued bearer token if an authorization code flow is used.
- All of the parameters issued in the response if an implicit flow is used.

Attribute sources

Both OpenID Connect Providers and Relying Parties can be configured to use an attribute source.

For an OpenID Connect Provider, this can be used instead of a mapping rule. However for an OpenID Connect Relying Party a mapping rule must still be present, this mapping rule is required to construct the principal used in the `iv-cred`.

For more information about attribute sources, see Managing attribute sources.

OpenID Connect mapping rules

Mapping rules allow users to customize the information that is propagated from an OpenID Connect Provider or what is consumed by a Relying Party.

These mapping rules can either be JavaScript, which is invoked internally via the STS, or the mapping can be performed externally via a HTTP request.

Import a mapping rule from another mapping rule

You can reuse mapping rules by importing a mapping rule from another mapping rule.

When you want to create a new mapping rule, or customize an existing mapping rule, you can reuse JavaScript code from a previously defined mapping rule. With this feature, you can define a mapping rule once and then reuse it in other mapping rules.

Use the function `importMappingRule()` to specify a mapping rule to import. For example, you can define a mapping rule that is called `Utility.js` that contains functions for obtaining an HTTP header and an HTTP cookie.

```
function getHeader(name) {
          // function for getting HTTP header
}

function getCookie(name) {
         // function for getting HTTP cookie
}
```

If you have another mapping rule that is called `Credential.js`, which also needs to obtain HTTP headers, use the following code to include the functions from the `Utility.js` mapping rule:

```
importMappingRule("Utility");
var host = getHeader("Host");
// do something with the host header
var sessionID = getHeader("PD-SESSION-ID");
// do something with the session ID
```

The function `importMappingRule()` accepts a list of mapping rule names and imports each of the mapping rules. For example:

```
importMappingRule("Utility","Credential","UserIdentity");
```

Alternatively, you can also make multiple calls to `importMappingRule()` within one script. For example:

```
importMappingRule("Utility");
importMappingRule("Credential");
importMappingRule("UserIdentity");
```

The JavaScript engine throws an error if you do not specify a mapping rule name, or if you specify the name of a mapping rule that does not exist.

Use the Local Management Interface (LMI) to view existing mapping rules that are defined on your system. Select **Federation > Global Settings > Mapping Rules**, or **AAC > Global Settings > Mapping Rules**.

Note:

On the LMI menu, the icon **Import** is for importing mapping rules into IBM Security Verify Access, not for importing a mapping rule into an existing mapping rule. Use the **Edit** icon to add the `importMappingRule()` function to an existing mapping rule.

Managing Distributed Session Cache

In a clustered appliance environment, session information is stored in the Distributed Session Cache. To work with these sessions, use the Distributed Session Cache management page.

About this task

The Distributed Session Cache feature replaces the Session Management Server. The Session Management Server (SMS) is not supported on IBM Security Verify Access for Web Version 8 and later.

Procedure

1. From the top menu, select the menu for your activation level.
    - **Web** > **Manage** > **Distributed Session Cache**
    - **AAC** > **Global Settings** > **Distributed Session Cache**
    - **Federation** > **Global Settings** > **Distributed Session Cache**

   All replica set names and the number of sessions in each replica set are displayed.

2. You can then view the replica set server list and manage sessions in a particular replica set.
   a) To view a list of the servers that are registered with a replica set, select the replica set and then click **Servers**.
   b) To manage the sessions in a replica set, select the replica set and then click **Sessions**.

      Tip: Typically, the list of sessions contains many entries. You can locate a session or a user faster by using the filter in the upper left corner.

      Delete a specific session

      1) Select the session to delete.
      2) Click **Delete**.
      3) In the confirmation window, click **Delete Session**.

      Delete all sessions for a user

      1) Select any session for that user.
      2) Click **Delete**.
      3) In the confirmation window, click **Delete User**.

Managing server connections

To access data from outside of your appliance, you must define a server connection.

Before you begin

Obtain the connection information for an existing LDAP database server.

About this task

With a Federation module activated, you can create server connections to an LDAP data source. You can have multiple servers for an LDAP connection.

Note: Even though other server connection types are available to select in the local management interface, such as DB2, only the LDAP server connection is used by Federation module.

If you also have the Advanced Access Control module activated, you can create any of the server connection types. See Managing server connections.

Procedure

1. Log in to the local management interface.
2. Click **Federation**.
3. Under **Global Settings**, click **Server Connections**.

4. Take one of the following actions:

   Filter server connections:

   a. In the Quick Filter field, type one or more characters. For example, enter g to search for all server connection names that contain g or G.
   b. Press Enter.

   Add a server connection:

   a. Click the drop-down button.
   b. Select **LDAP**.
   c. Complete the properties for the new server connection. See "Server connection properties" on page 283. Look specifically for the LDAP properties.

   Modify an existing server connection:

   a. Select a server connection.
   b. Click the edit icon .
   c. Complete the properties for the server connection. See "Server connection properties" on page 283.

   Delete a server connection:

   Note: Be careful about removing a server connection that is in use.

   a. Select a server connection.
   b. Click the delete icon .
   c. Click **Delete** to confirm the deletion.

What to do next

After you define a server connection to an LDAP data source, you can create an attribute source that looks up information from the LDAP server.

Server connection properties

To access a data source outside of the appliance, define the properties of the server.

The Server Connection properties table describes the properties on the **Server Connections** panel for the Advanced Access Control and Federation module activation levels.

- **Advanced Access Control**: Configure LDAP, database, web service, or Cloud Identity server connections so that you can set up policy information points. You can configure any of the server connection types.
- **Federation**: Configure an LDAP server as an attribute source for attribute mapping. Federation does not configure any of the other database server connection types.

Table 124. Server Connection properties

Property	Description
Name	Specifies the name for the server connection. Ensure that the name is unique. Select this name when you define the policy information point. **Note:** The server connection name must begin with an alphabetic character. Do not use control characters, leading and trailing blanks, and the following special characters ~ ! @ # $ % ^ & * ( ) + | ` = \ ; " ' < > ? , [ ] { } / anywhere in the name.

Table 124. Server Connection properties (continued)

Property	**Description**
Description	Describes the server connection. This property is optional.
Type	Shows the server connection type. (Read only)
JNDI ID (Oracle, DB2, PostgreSQL only)	Specifies the JNDI ID that the server uses. Ensure that the ID is unique. Use only alphanumeric characters: a-b, A-B, 0-9
Server name (Oracle, DB2, PostgreSQL, SMTP only)	Specifies the name or IP address for the server.
Port (Oracle, DB2, PostgreSQL, LDAP, SMTP only)	Specifies the port number where the connection to the server can be made.
URL (Web Service only)	Specifies the URL where the connection to the server can be made.
User name (Oracle, DB2, PostgreSQL, SMTP, and Web Service only)	Specifies the user name that has the correct permissions to access the resources.
Password (Oracle, DB2, PostgreSQL, SMTP, and Web Service only)	Specifies the password to access the server.
SSL	Specifies whether SSL is used for connecting to the server. Select **True** or **False**. The default value is **True**.
Driver type (Oracle only)	Specifies the driver type. Select **Thin** or **OCI**. The default value is **Thin**.
Service name (Oracle only)	Specifies the name of the service.
Database name (DB2, PostgreSQL only)	Specifies the name of the database.
Host name (LDAP only)	Specifies the host name or IP address of the LDAP server.
Bind DN (LDAP only)	Specifies the LDAP distinguished name (DN) that is used when binding, or signing on, to the LDAP server. **Note:** If this value is set to `"anonymous"`, the appliance uses an anonymous bind to the LDAP directory server. Typically the **bind-dn** has significant privileges so that it can be used to modify LDAP registry entries, such as creating users and resetting passwords via pdadmin or the Registry Direct Java API. Using an anonymous connection to LDAP typically comes with very limited access, perhaps at most search and view of entries, at the least no access at all. If anonymous access has sufficient privileges, then it might be usable for the WebSEAL level of access on users and groups. This access includes the permission for a user to change password if `"bind-auth-and-pwdchg = yes"` is set (`"ldap.bind-auth-and-pwdchg = true"` for Registry Direct Java API).
Bind Password (LDAP only)	Specifies the password for the LDAP bind DN. **Note:** If bind DN (`bind-dn`) is set to `anonymous`, you can use any non-empty string as the value of bind password (`bind-pwd`).
Administration hostname (Cloud Identity only)	Specifies the administration hostname of the Cloud Identity subscription.
Client ID (Cloud Identity only)	Specifies the client ID of an API Client on Cloud Identity.

Table 124. Server Connection properties (continued)

Property	Description
Client Secret (Cloud Identity only)	Specifies the client secret of an API Client on Cloud Identity.
SSL Truststore (LDAP, Web Service, and Cloud Identity only)	Specifies the truststore that verifies the credentials.
SSL Mutual Authentication Key (LDAP, Web Service, and Cloud Identity only)	Label of the client certificate to be presented when connecting to the LDAP. This property is sourced from SSL Truststore. **Note:** This field is required only if mutual SSL authentication is required by the server.

Note: For information on SSL configuration, see Configuring SSL connections.

The properties in the following table are connection manager properties. The defaults that are listed are the current known defaults. All tuning properties are optional.

Table 125. Tuning properties

Property	Description
Aged timeout (seconds) (Oracle, DB2, PostgreSQL only)	Specifies the amount of time, in seconds, before a physical connection is discarded by pool maintenance. Specify `-1` to disable this timeout. The default is `-1`.
Connection timeout (seconds)	Specifies the amount of time, in seconds, after which a connection times out. For Oracle, DB2, PostgreSQL, and SMTP, specify `-1` to disable this timeout. The default is `30` seconds. For LDAP, specify only integers, `1` or greater. The default is `120` seconds.
Max Idle Time (seconds) (Oracle, DB2, PostgreSQL only)	Specifies the maximum amount of time, in seconds, after which an unused or idle connection is discarded during pool maintenance. Specify `-1` to disable this timeout. The default is `1800` seconds.
Max Idle Time (seconds) (LDAP only)	Specifies the amount of time, in seconds, after which an established connection is discarded as idle. Set this to a value lower than the connection idle timeout on the LDAP server. **Note:** This is only applicable for performing Attribute Mapping from an LDAP server.
Reap time (seconds) (Oracle, DB2, PostgreSQL only)	Specifies the amount of time, in seconds, between runs of the pool maintenance thread. Specify `-1` to disable pool maintenance. The default is `180` seconds.
Max pool size (Oracle, DB2, PostgreSQL only)	Specifies the maximum number of physical connections for a pool. Specify `0` for unlimited. The default is `50`.
Max pool size (LDAP only)	Specifies the maximum number of connections that are pooled. **Note:** This is only applicable for performing Attribute Mapping from an LDAP server.
Min pool size (Oracle, DB2, PostgreSQL only)	Specifies the minimum number of physical connections to maintain in a pool. The aged timeout can override the minimum.

Table 125. Tuning properties (continued)

Property	Description
Purge policy (Oracle, DB2, PostgreSQL only)	Specifies which connections to delete when a stale connection is detected in the pool. Select from the following options: **Entire pool** When a stale connection is detected, all connections in the pool are marked stale, and when no longer in use, are closed. This is the default option. **Failing connection only** When a stale connection is detected, only the connection that was found to be bad is closed. **Validate all connections** When a stale connection is detected, connections are tested and the ones that are found to be bad are closed.
Max connections per thread (Oracle, DB2, PostgreSQL only)	Specifies the limit of open connections on each thread.
Cache connections per thread (Oracle, DB2, PostgreSQL only)	Specifies the number of cache connections for each thread.

Point of contact profiles

Use the local management interface to work with your point of contact profiles.

You can perform the following point of contact profile tasks:

- "Creating a point of contact profile" on page 286
- "Updating or viewing a point of contact profile" on page 287
- "Deleting a point of contact profile" on page 288
- "Setting a current point of contact profile" on page 288

Creating a point of contact profile

Create a point of contact server profile to capture the information needed for the runtime to communicate with the point of contact server.

About this task

You can create point of contact profiles with the Federation module or the Advanced Access Control module.

Three point of contact profiles provided by Security Verify Access are ready for use.

When you want to create your own profile that is similar to an existing one, use **Create Like** to save time. If you do not want to reuse any of the existing specifications, create a brand new one with **Create**. The details are in the following procedure.

Procedure

1. From the local management interface, select **Federation** or **AAC**. Then, **Global Settings** > **Point of Contact**.

   A list of point of contact server profiles displays. The list includes three preconfigured profiles and any other custom profiles that you created.

2. Take one of the following actions:
    - Click **Create** to create a custom point of contact profile.
    - Select a profile from the list and click **Create Like** to start with values similar to an existing profile.
3. On the Profile Name page, enter the name of the profile. The first character of the profile name must be alphanumeric. The maximum number of characters is 200.
4. Optional: Enter a description.
5. Specify the parameter information:
    - Enter the information on each tabbed page, and click **Next**.
    - In the Callback Parameters section on each page, click **Create** to open a window to add a set of parameter name and value pairs. Click **Save** when complete.
    - Add as many parameters as you need. The **Value** field might be empty for some parameters.
    - To delete a parameter name from the list, select the parameter and click **Delete**.
6. At the Summary page, if everything is correct, click **Finish**.
7. Deploy the pending changes.

What to do next

- See "Callback parameters and values" on page 288 for more information.
- You might want to change the current point of contact profile. See "Setting a current point of contact profile" on page 288.

Updating or viewing a point of contact profile

Update or view a point of contact server profile.

About this task

You cannot update the preconfigured point of contact profiles.

Procedure

1. From the local management interface, select **Federation** or **AAC**. Then, **Global Settings** > **Point of Contact**.

    A list of point of contact server profiles displays.
2. Perform one of the following actions:
    - Update
        a. Select a profile from the list that is not a preconfigured profile and click **Update** to change the configuration details.
        b. Click **Next** to see each page and make updates if necessary.
        c. On the Summary page, click **Finish** to save your changes.
        d. Deploy the changes
    - View
        a. Select a profile from the list and click **Properties** to look at the configuration details without making updates.
        b. Click on each tab to see the information.
        c. Click **OK** when finished.

What to do next

See "Callback parameters and values" on page 288 for more information about the properties.

Deleting a point of contact profile

Use the local management interface to remove a point of contact profile.

About this task

You cannot delete the following profiles:

- A preconfigured point of contact profile.
- A profile that is set as the current profile. Select another profile as the current one, if necessary.

See "Setting a current point of contact profile" on page 288.

Procedure

1. From the local management interface, select **Federation** > **Global Settings** > **Point of Contact** or **AAC** > **Global Settings** > **Point of Contact**.

   A list of point of contact server profiles displays.
2. Select a profile from the list, that is not a preconfigured profile, and click **Delete**.

   The details of the selected profile display.
3. Review the profile to ensure that it is the one you want to delete.
4. Click **Finish**.
5. Click **OK** to confirm.
6. Deploy the change.

Setting a current point of contact profile

Set a point of contact profile as the current one so that the federation runtime communicates with the point of contact server using the correct set of specifications.

Procedure

1. From the local management interface, select **Federation** > **Global Settings** > **Point of Contact** or select **AAC** > **Global Settings** > **Point of Contact**.

   A list of point of contact server profiles displays. The list includes three preconfigured profiles and any other custom profiles that you created. The green dot indicates the current profile.
2. To change the current profile, select the profile you want to use as the current one and click **Set As Current**.

   The current profile indicator displays next to the profile you selected.
3. Deploy the changes.

Callback parameters and values

Specify the callback parameters and values when you define a point of contact profile.

Sign In callbacks

fim.user.request.header.name

The name of the header that contains the user name of the user.

Data type: String

Example: `iv-user`

fim.attributes.response.header.name

The name of the header that contains the attributes of the user.

Data type: String

Example: `am-fim-eai-xattrs`

fim.groups.response.header.name
The name of the header that contains the groups of the user.

Data type: String

Example: `fim.groups`

fim.server.response.header.name
The name of the header that contains the hostname that authenticates the user.

Data type: String

Example: `fim.server`

fim.target.response.header.name
The name of the header that contains the redirect URL.

Data type: String

Example: `am-fim-eai-redir-url`

fim.user.response.header.name
The name of the header that contains the user name of the user.

Data type: String

Example: `am-fim-eai-user-id`

fim.user.session.id.response.header.name
The name of the header that contains the reverse proxy session ID of the user.

Data type: String

Example: `user_session_id`

fim.cred.response.header.name
The name of the header that contains the IVCred of the user.

Data type: String

Example: `am-fim-eai-pac`

url.encoding.enabled
Indicates whether the EAI header names and values are URL encoded. The default setting for this property is `false`. The EAI header names and values are not URL encoded.

Data type: Boolean

Example: `false`

Sign Out callbacks

fim.user.session.id.request.header.name
The name of the header that contains the reverse proxy session ID of the user.

Data type: String

Example: `user_session_id`

fim.user.request.header.name
The name of the header that contains the user name.

Data type: String

Example: `iv-user`

Local ID

fim.attributes.request.header.name
The name of the header that contains the attributes of the user.

Data type: String

Example: `fim.attributes`

fim.cred.request.header.name
The header that contains the IVCred of the user.

Data type: String

Example: `iv-creds`

fim.groups.request.header.name
The name of the header that contains the groups of the user.

Data type: String

Example: `iv-groups`

fim.user.request.header.name
The name of the header that contains the user name.

Data type: String

Example: `iv-user`

Authenticate

fim.user.request.header.name
The name of the header that contains the user name.

Data type: String

Example: `iv-user`

authentication.macros
A list of macros that defines contextual information to pass to the web reverse proxy login page. The macros you specify can customize an authentication login page for a specific service provider. For more information, see Customizing the SAML 2.0 login form.

Data type: String

Example: If an identity provider wants to display the provider ID and target URL of a partner, specify the following macros:

`%PARTNERID%,%TARGET%`

Access policies

You can use access policies to perform step-up and reauthentication during a single sign-on flow based on contextual information.

Access policies can be enforced at a federation or at API Protection for OAuth and OpenID Connect. The following list shows some example scenarios where access policies could be used.

- Restrict single sign-on access to applications based on the user and group membership.
- Restrict single sign-on access to applications based on devices, locations, and time.
- Require more authentication steps for single sign-on access to sensitive applications. Examples include re-authentication through an SMS one-time password, or confirmation of a push notification to a mobile device.
- Enforce user authentication requirements as demanded by an application, through a service provider, to grant single sign-on access.

Access policies can take contextual information as input:

- User information, such as user, groups, attributes
- Request information, such as HTTP headers, HTTP parameters, and cookies
- Single sign-on context, such as federation, partner, and authentication request. For OAuth and OpenID Connect the context includes Client ID, scope, response type, and other attributes.

Based on the contextual information, the administrator can choose from the following actions:

Allow
The user is allowed single sign-on access.

Deny
The user is denied single sign-on access.

Challenge
The user must complete a challenge before single sign-on access can proceed.

Access policies are defined as JavaScript. See "Access policy development" on page 292.

After an access policy is defined, it can be applied, used, and enforced on the following types of deployments.

- SAML 2.0 identity provider federation
- SAML 2.0 service provider partner to an identity provider federation
- OpenID Connect and API Protection Definition

Access policies cannot be applied or used by the following deployments.

- SAML 2.0 service provider federation
- SAML 2.0 identity provider partner to a service provider federation
- OpenID Connect and API Protection Client
- OpenID Connect Relying Party

For more information, see "Creating an access policy" on page 291.

Creating an access policy

You can create an access policy in JavaScript and then use the local management interface to deploy it.

Before you begin

Before you begin, ensure that you understand the following concepts.

- The business requirements or scenarios for the access policy.
- The types of Security Verify Access deployments that can enforce and use access policies.

For more information, see "Access policies" on page 290.

Procedure

1. Create the policy by writing JavaScript that enforces the requirements.

   See "Access policy development" on page 292.
2. Use the **Access Policies** menu in the local management interface to add the policy to your deployment.

   See "Managing access policies" on page 298.

3. Enable access policies for your deployment, and apply the necessary access policy.

   Follow the instructions for your type of deployment.

   - SAML 2.0 identity provider federation

     Use the local management interface **Federation** > **Manage** > **Federations** wizard to enable access policies, and select a policy to assign to the federation. See Creating and modifying a federation.
   - SAML 2.0 service provider partner to an identity provider federation

     Use the local management interface **Federation** > **Manage** > **Federations** wizard to enable access policies, and select a policy to assign to the partner. See Managing federation partners.

     Note: If you enable access policies on the partner, and select a policy, the partner policy takes precedence over any policy that is assigned to the federation. If you do not enable access policies on the partner, access policies that are enabled for the federation are still enforced.
   - OpenID Connect and API Protection Definition

     When you create or manage an API Protection Definition, you can choose to specify an access policy. See Creating an API protection definition.

Access policy development

You can use JavaScript to define and develop access policies.

Access policies are used to decide whether a user is allowed access to a single sign-on federation. Access policies return a decision of either Allow, Deny, or Challenge.

To write an access policy in JavaScript, use the Java classes, methods, and handlers that are supplied in Security Verify Access. To view the Javadoc, use the local management interface.

1. Select **System** > **File Downloads** > **`federation`** > **doc**
2. Access `ISAM-javadoc.zip`.

   Expand the Javadoc to view the relevant packages. For example:

   - com.ibm.security.access.policy
   - com.ibm.security.access.policy.decision
   - com.ibm.security.access.policy.saml20
   - com.ibm.security.access.policy.user

Allow

Use the allow decision to allow the single sign-on flow to continue if the requirement is met. The following example code shows a simple access policy that does not check any condition or requirement.

```
importClass(Packages.com.ibm.security.access.policy.decision.Decision);
var decision = Decision.allow();
context.setDecision(decision);
```

Another example is to allow the single sign-on flow to continue if the `username` equals `testuser`.

```
importClass(Packages.com.ibm.security.access.policy.decision.Decision);
importClass(Packages.com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler);
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.utilities);

//Retrieve the user context
var user = context.getUser();
//Retrieve the username
var username = user.getUsername();

if (username == "testuser"){
//Check the condition is username = testuser
       var decision = Decision.allow();
       context.setDecision(decision);
}
```

```
else{
       //If username is not testuser then deny the SSO flow.
       var handler = new HtmlPageDenyDecisionHandler();
       handler.setMacro("@MESSAGE@", JSON.stringify("Sorry "+username+ " is not allowed to run
     a successful Single Sign on flow"));
       var decision = Decision.deny(handler);
       context.setDecision(decision);
}
```

Challenge

Use the challenge decision to force the user to complete an action before the single sign-on flow can proceed.

The action might be to be redirected to a service that is running out of the Security Verify Access appliance, or to an HTML page that is provided by Security Verify Access, or to a custom HTML page, by setting a `pageid`. When you redirect to an HTML page, you can set macros to display data on the page.

The challenge decision can result in one of the following actions.

- HTMLPage Challenge

  This decision results in the display to the user of a default HTML page or a custom HTML. The default page is present under `/access_policy`, and is called `challenge_decision.html`.

  To challenge with a custom page, call the `setPageId` function with the path where the page is uploaded in Template Files.

  ```
  setPageId("/access_policy/Challenge_User.html");
  ```

  Macros can be set and retrieved by using the `setMacro` function.

  ```
  setMacro("@MESSAGE@", "Challenge Decision");
  ```

  When a challenge decision is called during a single sign-on operation, the single sign-on operation halts for the challenge to be completed. When the challenge is completed, the single sign-on operation must resume. To resume the operation, the page must be a `POST` operation on the `@ACTION@` macro, which resumes the flow.

- Redirect Challenge

  This decision results in an HTTP redirect to an external or third-party service. The user must complete the challenge before single sign-on can resume.

  In this scenario, the single sign-on flow is halted when a redirect challenge decision is initiated by the third-party server. The third party must be told which URL to redirect the user to, when the user successfully completes the challenge. Following is example code for a redirect challenge.

```
setRedirectUri("https://www.service.ibm.com/isam/service&redirectUri=https://www.myidp.ibm.com/
isam@ACTION@");
```

The URL `https://www.service.ibm.com/isam/service` is the third-party application or service that sends a challenge to the user. In this example, `https://www.myidp.ibm.com/isam` is the point of contact for the identity provider federation or OpenID Provider, and `@ACTION@` indicates the endpoint to access to resume the single sign-on flow. The Security Verify Access runtime server populates the value for the `@ACTION@` macro.

Deny

The Deny decision can result in one of the following actions.

- HTMLPage Deny

  This decision results in a default HTML page or a custom HTML page that is displayed to the user. The default page, `deny_decision.html`, is located under `/access_policy`.

  To deny with a custom page, call the `setPageId` function with the path where the page is uploaded in Template Files.

  ```
  setPageId("/access_policy/Deny_User.html");
  ```

  You can use the `setMacro` function to set and retrieve macros.

  ```
  setMacro("@MESSAGE@", "Deny Decision");
  ```

- Redirect Deny

  This decision results in an HTTP redirect to an external or third-party service. Example code for a redirect is as follows.

  ```
  setRedirectUri("https://www.denyService.com");
  ```

  The URI `https://www.denyService.com` is the third-party application to which the user is redirected.

User context example for access policy

You can specify an access policy that makes access decisions based on context that is obtained from the user information.

User context contains, users, groups, and attributes information. The following example access policy makes an access decision based on user context.

```
var userJSON = (function() {
       var user = context.getUser();
       var userReturn = {};

       var groupsJSON = (function() {
              var groupsReturn = [];
              var groups = user.getGroups();

              for (var it = groups.iterator(); it.hasNext();) {
                     var group = it.next();
                     var groupName = group.getName();
                     groupsReturn.push("" + groupName);
              }

              return groupsReturn;
       })();

       var attributesJSON = (function() {
              var attributesReturn = {};
              var attributes = user.getAttributes();

              for (var it = attributes.iterator(); it.hasNext();) {
                     var attribute = it.next();
                     var attributeName = attribute.getName();
                     var attributeValue = attribute.getValue();

                     attributesReturn["" + attributeName] = "" + attributeValue;
              }

              return attributesReturn;
       })();

       userReturn["username"] = "" + user.getUsername();
       userReturn["groups"] = groupsJSON;
       userReturn["attributes"] = attributesJSON;

       return userReturn;
})();
```

Request context example for access policy

You can specify an access policy that makes access decisions based on context that is obtained from the request.

Requests can contain headers, cookies, and parameters. The following example uses the request context to make an access decision.

```
//Retrieve request context
var requestJSON = (function() {
       var request = context.getRequest();
       var requestReturn = {};

       var headersJSON = (function() {
              var headersReturn = {};
              var headerNames = request.getHeaderNames();

              for (var it = headerNames.iterator(); it.hasNext();) {
                     var headerName = it.next();
                     var headerValue = request.getHeader(headerName);

                     headersReturn["" + headerName] = "" + headerValue;
              }

              return headersReturn;
       })();

       var cookiesJSON = (function() {
             var cookiesReturn = {};
             var cookies = request.getCookies();

             for (var it = cookies.iterator(); it.hasNext();) {
                     var cookie = it.next();
                     var cookieComment = cookie.getComment();
                     var cookieDomain = cookie.getDomain();
                     var cookieHttpOnly = cookie.isHttpOnly();
                     var cookieMaxAge = cookie.getMaxAge();
                     var cookieName = cookie.getName();
                     var cookiePath = cookie.getPath();
                     var cookieSecure = cookie.isSecure();
                     var cookieValue = cookie.getValue();
                     var cookieVersion = cookie.getVersion();

                     cookiesReturn["" + cookieName] = {
                            comment: "" + cookieComment,
                            domain: "" + cookieDomain,
                            httpOnly: cookieHttpOnly,
                            maxAge: cookieMaxAge,
                            path: "" + cookiePath,
                            secure: cookieSecure,
                            value: "" + cookieValue,
                            version: cookieVersion
                     };
              }

              return cookiesReturn;
       })();

       var parametersJSON = (function() {
              var parametersReturn = {};
              var parameterNames = request.getParameterNames();

              for (var it = parameterNames.iterator(); it.hasNext();) {
                     var parameterName = it.next();
                     var parameterValue = request.getParameter(parameterName);

                     parametersReturn["" + parameterName] = "" + parameterValue;
              }

              return parametersReturn;
       })();

       requestReturn["headers"] = headersJSON;
       requestReturn["parameters"] = parametersJSON;

       return requestReturn;
})();
```

SAML 2.0 protocol context example for access policy

You can specify an access policy that makes access decisions based on context that is obtained from the protocol.

For SAML 2.0, the protocol context includes federation information, partner information, and the authentication request. The following policy makes an access decision based on the protocol context.

```
//Retrieve protocol context
var protocolContextJSON = (function() {
       var protocolContext = context.getProtocolContext();
       var protocolContextReturn = {};
       protocolContextReturn["request"] = "" + protocolContext.getAuthnRequest();
       protocolContextReturn["FederationId"] = "" + protocolContext.getFederationId();
       protocolContextReturn["PartnerId"] = "" + protocolContext.getPartnerId();
       protocolContextReturn["FederationName"] = "" + protocolContext.getFederationName();
       protocolContextReturn["PartnerName"] = "" + protocolContext.getPartnerName();
       return protocolContextReturn;
})();
```

An example of using SAML 2.0 protocol context to decide whether to allow or deny based on the partner name is as follows.

```
importClass(Packages.com.ibm.security.access.policy.decision.Decision);
importClass(Packages.com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler);

var protocolContext = context.getProtocolContext();

if (protocolContext.getPartnerName() != "SP Company"){
       var decision = Decision.allow();
       context.setDecision(decision);
}
else{
       var handler = new HtmlPageDenyDecisionHandler();
       handler.setMacro("@MESSAGE@", "Sorry "+protocolContext.getPartnerName()+ " is not allowed
              to run a successful Single Sign on flow");
       var decision = Decision.deny(handler);
       context.setDecision(decision);
}
```

OAuth and OpenID Connect protocol context example for access policy

You can specify an access policy that makes access decisions based on context that you obtained from the OAuth and OpenID Connect protocol.

Some examples scenarios that make use of an access policy with an OAuth and OpenID Connect deployment are as follows.

- An access policy performs extra authentication:
  - For a particular client
  - For a certain flow based, on `response_type` requested.
  - When a specific `scope` is requested.
- An access policy decided to re-authenticate the user when the last authentication time is greater than the `max_age` that was requested.

Following is an example of protocol context for OpenID Connect.

```
//Retrieve protocol context
var protocolContextJSON = (function() {
       var protocolContext = context.getProtocolContext();
       var protocolContextReturn = {};
       protocolContextReturn["request"] = "" + protocolContext.getAuthenticationRequest();
       protocolContextReturn["ClientId"] = "" + protocolContext.getClientId();
       protocolContextReturn["ClientName"] = "" + protocolContext.getClientName();
       protocolContextReturn["DefinitionId"] = "" + protocolContext.getDefinitionId();
       protocolContextReturn["DefinitionName"] = "" + protocolContext.getDefinitionName();
       return protocolContextReturn;
})();
```

An example of using OpenID Connect 2.0 protocol context to make a decision to allow or deny based on the client ID name is as follows.

```
importClass(Packages.com.ibm.security.access.policy.decision.Decision);
importClass(Packages.com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler);
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.utilities);

//This access policy denies successful Single Sign On, if clientid = clientID
var protocolContext = context.getProtocolContext();

if (protocolContext.getClientId() == "clientID")
{
       var handler = new HtmlPageDenyDecisionHandler();
       //Setting the macro with all the context information,
       // make sure that the /access_policy/deny_decision.html is modified to print the macro.
       handler.setMacro("@MESSAGE@", "Single Sign On cannot be completed by the
              following clientId : "+protocolContext.getClientId());
       var decision = Decision.deny(handler);
       context.setDecision(decision);
}
else
{
       var decision = Decision.allow();
       context.setDecision(decision);
}
```

Session context example for access policy

You can specify an access policy that makes access decisions based on context that is obtained from the session.

```
var session = context.getSession();
//If a session attribute called 'text' exists the following function will retrieve its value
var sessionData = session.getAttribute("text");
```

Reauthentication example for access policy

Access policies can be used to do reauthentication. Following is an example of an access policy that implements reauthentication.

Note: Reauthentication in an access policy is supported only by performing a Redirect Challenge to the required authentication service.

```
// max_age represents how long a user session should be active
// authenticationTime represents when the user first logged into the authentication service
if (max_age < authenticationTime){
     // Reauthenticate the user using a username password policy
     // Once the username password policy is executed, the authenticationTime will be refreshed and
     // the "if" condition will not hold good; hence the user will be allowed to run the single sign-on
     var handler = new RedirectChallengeDecisionHandler();
     handler.setRedirectUri("https://www.myidp.example.com/isam/sps/authsvc?PolicyId=
        urn:ibm:security:authentication:asf:password&Target=https://www.myidp.example.com/isam@ACTION@");
     var decision = Decision.challenge(handler);
     context.setDecision(decision);
}
else{
     var decision = Decision.allow();
     context.setDecision(decision);
}
```

Template files for access policies

You can use template files to build your access policies.

You can access the template files from the local management interface.

1. Select **Federation** > **Template Files**
2. Expand the `access_policy` entry.

The following template files are provided.

Table 126. Access Policy templates

Template	Description
challenge_decision.html	An HTML page that instructs the user to complete a challenge in order to gain access. The challenge is specified by the value you put into the @ACTION@ macro.
deny_decision.html	An HTML page that informs the user that access is denied.
server_error.html	An HTML page that you can use to display an error message to the user. You can assign values for the following macros: • @REQ_ADDR@ • @TIMESTAMP@ • @EXCEPTION_MSG@ • @EXCEPTION_STACK@

Managing access policies

You can use the local management interface to manage access policies.

Before you begin

Ensure you understand how to develop and use access policies for federation single sign-on. For more information, see "Access policies" on page 290.

Procedure

1. In the local management interface, select **Federation** > **Global Settings** > **Access Policies**.
2. Select the action you want to complete.

   Add

   Enter the JavaScript code that you want to include in this policy. Enter a name (string) for the policy. For Type, select JavaScript. For Category, you can specify any string. Save the entry.

   Import

   Enter a name for the policy and select a type. Use the Browse window to select a JavaScript file to import. Click **OK** to import the file.

   Edit

   Select a policy from the Access Policies list. Click **Edit**. Change the JavaScript as needed and click **OK**.

   Delete

   Select a policy from the Access Policies list. Click **Delete** to remove the policy.

   Export

   Select a policy from the Access Policies list. Click **Export** to save the policy to disk.

   Replace

   Select a policy from the Access Policies list. Click **Replace** to replace the policy. Click Browser to locate the JavaScript file that contains the policy that you want to use instead of the existing

JavaScript for the policy that you selected. Click **OK** to replace the existing JavaScript in the selected policy with the JavaScript from the selected file.

3. When prompted, deploy your changes.

What to do next

Create or modify a federation or federation partner to use the access policy. See "Creating an access policy" on page 291.

Sample file for Access Policies

Use the Access Policies samples as a template and modify it to suit your needs.

The access policy samples help you to get started with access policies. These samples assume that the federation, partner, and reverse proxy are configured with the correct junction, federation, and partner name.

```
importClass(Packages.com.ibm.security.access.policy.decision.Decision);
importClass(Packages.com.ibm.security.access.policy.decision.HtmlPageDenyDecisionHandler);
importClass(Packages.com.ibm.security.access.policy.decision.RedirectDenyDecisionHandler);
importClass(Packages.com.ibm.security.access.policy.decision.HtmlPageChallengeDecisionHandler);
importClass(Packages.com.ibm.security.access.policy.decision.RedirectChallengeDecisionHandler);
importPackage(Packages.com.tivoli.am.fim.trustserver.sts.utilities);

//Set promptTOTP = true if the user must be prompted with TOTP during a single sign on flow.
var promptTOTP = false;
if (promptTOTP){
/*
* We are using the TOTP policy that is bundled with the Advanced Access Control activation.
* The isamcfg tool must be configured with the right junction name.
*/
 //Retrieve user context
 var user = context.getUser();
 //Check the various authenticationTypes performed by the user
 var authenticationTypesAttribute = user.getAttribute("authenticationTypes");
 if (authenticationTypesAttribute != null && authenticationTypesAttribute.getValues().
contains("urn:ibm:security:authentication:asf:totp")){
  /*
  * If authenticationTypesAttribute is not null, we check if the user has performed TOTP,
  * if yes the user is allowed to continue with the Single Sign on.
  */
  context.setDecision(Decision.allow());
 }
 else{
  /*
   * If authenticationTypesAttribute is null, or the user has not performed TOTP, the
   * user is challenged with a TOTP authentication.
   * This is done by using a RedirectChallengeDecision. The RedirectChallengeDecision
   * handler needs a redirect uri to which the user must be redirected to. Below is the
   * API which does that.
   * handler.setRedirectUri("/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:
   * totp&Target=https://www.myidp.ibm.com/isam@ACTION@");
   * Notice the Uri, it invokes a TOTP policy that is available OOTB by activating the
   * Advanced Access Control, the other parameter which is sent is the Target, this is
   * the URL the user will be redirected to once the TOTP is completed.
   * The format of the URL is https://www.myidp.ibm.com/isam@ACTION@ , where
   * https://www.myidp.ibm.com/isam is the point of contact server for the federation
   * and @ACTION@ macro is the endpoint which needs to be accessed for the Single Sign On
   * flow to continue, since it was halted when the redirect challenge was initiated.
   */

  var handler = new RedirectChallengeDecisionHandler();
  /*
  * If a variable or a string needs to be logged into the trace.log use the
  * IDMappingExtUtils.traceString() function. To enable the trace, set the trace string to
  * com.tivoli.am.fim.*:ALL
  */
  IDMappingExtUtils.traceString("CHALLENGE WITH TOTP");
  handler.setRedirectUri("/sps/authsvc?PolicyId=urn:ibm:security:authentication:asf:totp\
&Target=https://www.myidp.ibm.com/isam@ACTION@");
  context.setDecision(Decision.challenge(handler));
 }
}

/*
* Set checkGroupMembership = true if the user is allowed to perform single sign on flow based on
* group membership.
*/
var checkGroupMembership = false;
if (checkGroupMembership){
 //Retrieve user context
 var user = context.getUser();
 //Check if the user belongs to the "SecurityGroup"
 var group = user.getGroup("SecurityGroup");
 //If the user belongs to the group, else Deny
 if ( group != null){
  context.setDecision(Decision.allow());
 }
 else{
  /*
   * If the user does not belong to the group, the single sign on flow is aborted. A
         * HtmlPageDenyDecision is used to deny the user from performing SSO.
   * The HtmlPageDenyDecision throws an OOTB HTML Deny page, which is located at
         * /access_policy/deny_decision.html or we could set a custom page using setPageId().
   * A custom macro could be sent to display a custom error messages using setMacro().
   *
   * var handler = new HtmlPageDenyDecisionHandler();
   * handler.setPageId("/access_policy/custom_deny_decision.html");
   * handler.setMacro("@MESSAGE@","This is a custom deny page");
   *
   * Make sure that the following page exists /access_policy/custom_deny_decision.html,
   * a macro can be set to so that it can be retrieved from the template page.
   *
   * In the above example a @MESSAGE@ macro is set, this can be retrieved in the
   * /access_policy/custom_deny_decision.html page using the following code snippet.
   *
   * <%templateContext.response.body.write(templateContext.macros["@MESSAGE@"]);%>
   *
   * <div class="pageContent">
   *  <div class="errorMessage"><%templateContext.response.body.write(templateContext
   * .macros["@MESSAGE@"]);%></div>
   * </div>
   *
   */
  var handler = new HtmlPageDenyDecisionHandler();
  handler.setMacro("@MESSAGE@", "This user does not belong to the required group and is\
not allowed to preform sso");
  context.setDecision(Decision.deny(handler));
 }

}
```

With the Access Policies above, there are samples for each of the following activities:

- Redirecting to another authentication provider, and the pattern for returning to the SSO
- Checking credential attributes
- Checking group membership
- Sending a HTML page
- Setting a macro for a page
- Sending a redirect
- Denying a request
- Allowing a request

Runtime monitoring using Prometheus

Runtime can be configured to provide a `/metrics` REST interface from which you can access all metrics that are emitted by the Runtime. The default format for responses to requests to `/metrics` is a text format that is compatible with Prometheus.

This is controlled by the advance tuning parameter `runtime_profile.enable.monitor`. To enable, set the parameter to `true` and deploy pending changes. To disable, set the `runtime_profile.enable.monitor` to `false`

The monitoring endpoint is unprotected. If the service needs to be protected, it needs to be done with a WebSEAL junction.

Use `/metrics` to access the monitoring data.

Chapter 10. DB2 HVDB High Availability Disaster Recovery (HADR) guideline

HADR provides a high availability solution for both partial and complete site failures.

HADR protects against data loss by replicating data changes from a source database, called the primary database, to the target databases, called the standby databases. HADR supports up to three remote standby servers.

Description of the HADR Modes

HADR synchronization mode is controlled by the database configuration parameter **`hadr_syncmode`**.

See DB2 v1.0.1 Knowledge Center.

HADR provides four synchronization modes to suit a diverse range of operational environment. Database configuration parameter **`hadr_syncmode`** can be set to one of the following modes:

SYNC
: Transactions on primary commits only after relevant logs are written to disk on both primary and standby.

NEARSYNC
: Transactions on primary commits only after relevant logs are written to disk on primary and received into memory on standby.

ASYNC
: Transactions on primary commits only after relevant logs have been written to local disk and sent to standby.

SUPERASYNC
: Transactions on primary does not wait for replication of logs to the standby.

For SYNC and NEARSYNC modes, the primary waits for an `ack` message from the standby to confirm that the logs are received and written to disk on standby (SYNC mode) or are received on the standby (NEARSYNC mode).

For ASYNC mode, primary considers replication as done as soon as the logs are delivered to the TCP layer of the primary host machine. For SUPERASYNC mode, the primary log writing is independent of log replication.

SYNC and NEARSYNC modes are typically used on LAN. ASYNC and SUPERASYNC modes are typically used over WAN.

SYNC mode

SYNC mode provides the most secure data protection. Two on-disk copies of data are required for transaction commit.

The downside of this mode is the extra time for writing on standby and sending the `ack` message back to primary.

In SYNC mode, logs are sent to standby only after they are written to primary disk. Log write and replication events happen sequentially. The total time for a log write is the sum of the `primary_log_write`, `log_send`, `standby_log_write`, and `ack_message`. The cost of replication is significantly higher than other modes.

NEARSYNC mode

NEARSYNC mode is comparable to the SYNC, mode at significantly less cost.

Standby sends `ack` message as soon as it receives the logs in memory. It also sends logs to standby and writes logs to primary disk in parallel. On a fast network, log replication causes no or little overhead to primary log writing.

In NEARSYNC mode, you lose data if primary fails and the standby fails before it has a chance to write the received logs to disk. This is a rare "double failure" scenario. Thus NEARSYNC mode is a good choice for many applications, providing near synchronization protection at a far less performance cost.

ASYNC mode

In ASYNC mode, sending logs to standby and writing logs to primary disk are done in parallel.

ASYNC mode does not wait for `ack` messages from the standby, primary system throughput is minimum (log write rate, log send rate). ASYNC mode is applicable for WAN application. Network transmission delay does not impact performance in this mode. However, if the primary database fails, there is a higher chance that logs in transit are lost (not replicated to standby).

SUPERASYNC mode

In SUPERASYNC mode, log writing and replication are independent. HADR never enters peer state.

Log shipping only uses remote catchup state. Log writing is never slowed down. However, primary-standby log gap can grow. In a failover, data in the gap is lost. You must monitor the gap closely. This mode provides the least impact on primary, at the cost of the least data protection. It is typically used on unreliable networks.

While in other sync modes, a non-forced takeover is allowed only in peer state, where primary and standby log positions are close, in the SUPERASYNC mode, non-forced takeover is allowed in remote catchup state. If there is a large gap, the takeover takes a long time because after stopping transactions on the primary, HADR is still required to ship all logs in the gap to the standby and replay them before takeover can complete. You must to check the gap before issuing a non-forced takeover in SUPERASYNC mode.

Choose a synchronization Mode

You can choose synchronization mode types for the IBM Security Verify Access Federation component.

SAML

SAML 2.0 Flow	Binding	NameID Management	Recommended replication mode	Comments
Single Sign-On (SSO)	HTTP POST	Email, Transient	NEARSYNC	If Single Log Out is not required, choose the SUPERASYNC mode.
	HTTP REDIRECT	Email, Transient	NEARSYNC	
	HTTP Artifact	Email, Transient	NEARSYNC	The Service Provider or Identity Provider must resolve the SAML Artifact from the Identity Provider or Service Provider. In case of a database failover during an SSO, the SAML message must be in standby for the Service Provider or Identity Provider to be able to resolve it.
	• HTTP POST • HTTP ARTIFACT • HTTP REDIRECT	Persistent	NEARSYNC	ALIAS_SVC_ALIASUSERPARTNER data is replicated in case of failover.

OpenID Connect (OIDC) or OAuth

OIDC Flow	Response type	Recommended replication mode	Comment
Authorization code flow	code	NEARSYNC	At authorization code flow, the Relying Party client is required to exchange an authorization code for a token. In case of failover, the Relying Party must get the authorization code resolved from the secondary database.
Implicit	• token • id_token	NEARSYNC	In Implicit flow, the refresh token is not generated. To improve performance, use the SUPERASYNC mode.
Hybrid	• code • token • id_token	NEARSYNC	At hybrid flow, Relying Party client is required to exchange an authorization code for a token. In case of failover Relying Party needs to get the authorization code resolved from a secondary database.

WS Federation Single Sign-On (WSFed SSO)

Recommended HADR mode: NEARSYNC.

Note: If the single log out feature is not required we can use the SUPERASYNC mode.

SAML 1.1

SAML 1.1 Flow	Binding	Recommended replication mode	Comment
Single Sign-On	HTTP POST	SUPERASYNC	
Single Sign-On	HTTP Artifact	NEARSYNC	The Service Provider or Identity Provider must resolve the SAML Artifact from the Identity Provider or Service Provider. In case of a database failover during an SSO, the SAML message must be in standby for the Service Provider or Identity Provider to be able to resolve it.

For more information on synchronization mode types for the IBM Security Verify Access Advanced Access Control component, see Choose a synchronization mode for the Advanced Access Control component.

Index