IdPMetadataProvider

Define a New Metadata Source

A great deal of functionality in the Shibboleth Identity Provider is driven from SAML metadata information. Metadata is provided to the IdP through Metadata Providers (yeah, we developers are pretty creative with our names). Metadata Providers are defined in the *$IDP_HOME/conf/relying-party.xml* file and are defined as follows.

For help with understanding and/or creating SP metadata, see the Metadata and MetadataForSP topics.

Define the Metadata Provider

The IdP may have one, logical, Metadata Provider per relying partying group (though multiple, individual, Metadata Providers can be chained together to create the logical Provider used).

Metadata Providers are defined with a `MetadataProvider` element appearing after all the `RelyingParty` elements in the *relying-party.xml* file. Like other components the specific type of Metadata Provider is identified by the `xsi:type` attribute. Metadata Providers must also have an `id` attribute which assigns a unique id to the provider. This is used for debugging and logging purposes. The following types are supported.

The default IdP configuration uses a chaining metadata provider. You should define any new metadata providers within that chaining provider.

About Reloading Metadata Providers

As of version 2.2, all metadata providers which fetch metadata from an external (to the IdP software) source have undergone a change in when this fetching is performed. Prior to version 2.2 metadata was fetched during the first profile request that followed the cached metadata's expiration. If the metadata document was large this caused a very noticeable, but inexplicable, delay in the request processing. As of version 2.2 all metadata reloading occurs in the background and does not affect request processing (i.e. no more pauses due to metadata reloading).

The background process used to download metadata is designed to prevent problems where cached metadata expires but the remote source can not be reached. The algorithm used to determine the next refresh cycle uses three configurable inputs in addition to `cacheDuration` attributes that may be found in the metadata document itself. These inputs are: *refreshDelayFactor*, *minRefreshDelay*, and *maxRefreshDelay*. The later two are easy enough to understand. *minRefreshDelay* represents the smallest amount of time between cycles. *maxRefreshDelay* represents the largest amount of time between cycles.

The *refreshDelayFactor* is a bit harder to understand, but if you're familiar with how DHCP leases work you'll be familiar with the concept. The next refresh cycle for metadata is determined with the following algorithm:

1. Determine the earliest date/time of all (`validUntil`) and (now + `cacheDuration`) and (now + *maxRefreshDelay*)
2. If that time is before now, schedule a refresh in (now + *minRefreshDelay*). Stop.
3. Compute the difference between now and time determined in step one.
4. Multiply that number by the *refreshDelayFactor* to get the delay interval
5. Schedule the next refresh in now + computed delay interval

By using the delay factor in this way the IdP can get in a few refreshes (assuming reasonable values for the three configuration options) before a metadata document expires. This helps protect against problems that might arise if the remote metadata source is unavailable for short periods of time. It does not protect against metadata sources that are down for long periods of time.

Chaining Metadata Provider

This Metadata Provider provides a means of logically combining other Metadata Provider definitions. When the IdP requests metadata from this type of provider the provider returns the response of the first metadata provider, in the chain, that provides a response. For example, if providers A, B, and C are defined in a chain and both B and C contain an entity descriptor for *urn:example.org:idp1*, which the IdP is searching for, only the entity descriptor from B will be returned as it will be the first provider to return a non-empty response. No attempt to merge metadata from various metadata providers is made.

This provider is defined by the element `<MetadataProvider xsi:type="ChainingMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attribute:

- **id** - provides a unique, amongst metadata providers, identifier that may be used to reference the provider

Example File Backed HTTP Metadata Provider Configuration

```
<!-- RelyingParty elements above this point -->
<MetadataProvider xsi:type="ChainingMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata"
                  id="MyMetadata">

    <MetadataProvider xsi:type="FilesystemMetadataProvider"
                      id="InternalMetadata"
                      metadataFile="/path/to/my/metadata-internal.xml" />

    <MetadataProvider xsi:type="FileBackedHTTPMetadataProvider"
                      id="External Metadata"
                      metadataURL="http://example.org/metadata.xml"
                      backingFile="/tmp/idp-metadata.xml" />

</MetadataProvider>
```

Filesystem Metadata Provider

The Filesystem Metadata Provider reads SAML 2 metadata from a file on the file system. Metadata is cached in memory for a period of time in order to improve performance. The metadata provider also monitors the file for changes and will reload the file upon detecting an update.

It is defined by the element `<MetadataProvider xsi:type="FilesystemMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attributes:

- **id** - provides a unique, amongst metadata providers, identifier that may be used to reference the provider
- **metadataFile** - the filesystem path to the metadata file

and the following optional attributes:

- **failFastInitialization** (added in v2.2) - a boolean flag indicating whether, during startup, the IdP should fail to start if there is a problem fetching metadata (default value: true)
- **requireValidMetadata** (added in v2.2) - a boolean flag that indicates that loaded metadata must be valid, according to the validUntil attributes found in the metadata (default value: true)
- **refreshDelayFactor** (added in v2.2) - an number between 0.0 and 1.0, exclusive, used to determine the next metadata refresh cycle based on the current metadata's cache expiration time (default value: 0.75)
- **minRefreshDelay** (added in v2.2) - a lower bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT5M)
- **maxRefreshDelay** (added in v2.2) - an upper bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT4H)

Setting the min and max refresh delay to the same value is a nonsensical configuration. Don't do it.

Example Filesystem Metadata Provider Configuration

```
<!-- RelyingParty elements above this point -->
<MetadataProvider xsi:type="FilesystemMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata"
                  id="MyMetadata"
                  metadataFile="/path/to/my/metadata.xml" />
```

File Backed HTTP Metadata Provider

The File Backed HTTP Metadata Provider works like the HTTP Metadata Provider, but stores metadata in a local temporary file upon successful retrieval. This protects against the case where metadata has been successfully fetched once but the remote server is offline during subsequent requests (even across IdP reboots). For users of Shibboleth 1.3 this Metadata Provider provides the same functionality as the common setup of a scheduled fetch via metadatatool and subsequent read for the metadata from the filesystem.

Note, as of version 2.2, this metadata provider supports deflate and gzip compression as well as conditional fetching of metadata based on Last-Modified and ETag headers.

It is defined by the element `<MetadataProvider xsi:type="FileBackedHTTPMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attributes:

- **id** - provides a unique, amongst metadata providers, identifier that may be used to reference the provider
- **metadataURL** - the URL of the metadata file
- **backingFile** - the filesystem path where a backup copy of the metadata file is stored

and the following optional attributes:

- **failFastInitialization** (added in v2.2) - a boolean flag indicating whether, during startup, the IdP should fail to start if there is a problem fetching metadata (default value: true)
- **requireValidMetadata** (added in v2.2) - a boolean flag that indicates that loaded metadata must be valid, according to the validUntil attributes found in the metadata (default value: true)
- **maintainExpiredMetadata** (deprecated in v2.2, use **requireValidMetadata** instead) - A boolean flag indicating whether metadata should be retained, and used, after it has expired and the defined URL can not be reached to retrieve fresh metadata, defaults to false
- **refreshDelayFactor** (added in v2.2) - an number between 0.0 and 1.0, exclusive, used to determine the next metadata refresh cycle based on the current metadata's cache expiration time (default value: 0.75)
- **minRefreshDelay** (added in v2.2) - a lower bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT5M)
- **maxRefreshDelay** (added in v2.2) - an upper bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT4H)
- **cacheDuration** (deprecated in v2.2, use **maxRefreshDelay** instead) - Maximum length of time, in seconds, metadata should be cached for, defaults to 2880. If the metadata file expressed a shorter duration, either through an explicit expiration date or by means of a cache duration, then the metadata specified duration will take precedence
- **disregardSslCertificate** (added in v2.2) - boolean flag indicating whether the servers SSL certificate should always be accepted regardless of whether its valid (defaults value: false)
- **requestTimeout** - Maximum length of time to wait for the remote server to finish its response given in XML duration notation (default value: PT5S). For versions prior to v2.2, use value in milliseconds.
- **proxyHost** (added in v2.2) - hostname of the HTTP proxy to use when fetching metadata
- **proxyPort** (added in v2.2) - port of the HTTP proxy to use when fetching metadata
- **proxyUser** (added in v2.2) - username used when connecting to the HTTP proxy to use when fetching metadata
- **proxyPassword** (added in v2.2) - password used when connecting to the HTTP proxy to use when fetching metadata
- **basicAuthUser** (added in v2.2) - HTTP BASIC authentication username used when connecting to the HTTP proxy to use when fetching metadata
- **basicAuthPassword** (added in v2.2) - HTTP BASIC authentication password used when connecting to the HTTP proxy to use when fetching metadata

Due to limitations with the Apache Commons HTTP Client 3.x API, all HTTP and File Backed HTTP metadata providers **MUST** have the same value configured for **disregardSslCertificate**. Failure to do so will result in unpredictable TLS certificate validation and hostname verification behavior amongst the providers. All providers in effect use the same TLS protocol socket factory.

Setting the min and max refresh delay to the same value is a nonsensical configuration. Don't do it.

Example File Backed HTTP Metadata Provider Configuration

```
<!-- RelyingParty elements above this point -->
<MetadataProvider xsi:type="FileBackedHTTPMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata"
                  id="MyMetadata"
                  metadataURL="http://example.org/metadata.xml"
                  backingFile="/tmp/idp-metadata.xml" />
```

HTTP Metadata Provider

The HTTP Metadata Provider reads metadata from an `http` or `https` scheme URL. Metadata is cached in memory for a period of time in order to improve performance. Deployers are strongly encourage to use the File Backed HTTP Metadata Provider described above instead of this provider.

Note, as of version 2.2, this metadata provider supports deflate and gzip compression as well as conditional fetching of metadata based on Last-Modified and ETag headers.

It is defined by the element `<MetadataProvider xsi:type="HTTPMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attributes:

- **id** - provides a unique, amongst metadata providers, identifier that may be used to reference the provider
- **metadataURL** - the URL of the metadata file

and the following optional attributes:

- **failFastInitialization** (added in v2.2) - a boolean flag indicating whether, during startup, the IdP should fail to start if there is a problem fetching metadata (default value: true)
- **requireValidMetadata** (added in v2.2) - a boolean flag that indicates that loaded metadata must be valid, according to the validUntil attributes found in the metadata (default value: true)
- **maintainExpiredMetadata** (deprecated in v2.2, use **requireValidMetadata** instead) - A boolean flag indicating whether metadata should be retained, and used, after it has expired and the defined URL can not be reached to retrieve fresh metadata, defaults to false
- **refreshDelayFactor** (added in v2.2) - an number between 0.0 and 1.0, exclusive, used to determine the next metadata refresh cycle based on the current metadata's cache expiration time (default value: 0.75)
- **minRefreshDelay** (added in v2.2) - a lower bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT5M)
- **maxRefreshDelay** (added in v2.2) - an upper bound on the frequency of metadata refresh cycles given in XML duration notation (default value: PT4H)
- **cacheDuration** (deprecated in v2.2, use **maxRefreshDelay** instead) - Maximum length of time, in seconds, metadata should be cached for, defaults to 2880. If the metadata file expressed a shorter duration, either through an explicit expiration date or by means of a cache duration, then the metadata specified duration will take precedence
- **disregardSslCertificate** (added in v2.2) - boolean flag indicating whether the servers SSL certificate should always be accepted regardless of whether its valid (defaults value: false)
- **requestTimeout** - Maximum length of time to wait for the remote server to finish its response given in XML duration notation (default value: PT5S). For versions prior to v2.2, use value in milliseconds.
- **proxyHost** (added in v2.2) - hostname of the HTTP proxy to use when fetching metadata
- **proxyPort** (added in v2.2) - port of the HTTP proxy to use when fetching metadata
- **proxyUser** (added in v2.2) - username used when connecting to the HTTP proxy to use when fetching metadata
- **proxyPassword** (added in v2.2) - password used when connecting to the HTTP proxy to use when fetching metadata
- **basicAuthUser** (added in v2.2) - HTTP BASIC authentication username used when connecting to the HTTP proxy to use when fetching metadata
- **basicAuthPassword** (added in v2.2) - HTTP BASIC authentication password used when connecting to the HTTP proxy to use when fetching metadata

Due to limitations with the Apache Commons HTTP Client 3.x API, all HTTP and File Backed HTTP metadata providers **MUST** have the same value configured for **disregardSslCertificate** . Failure to do so will result in unpredictable TLS certificate validation and hostname verification behavior amongst the providers. All providers in effect use the same TLS protocol socket factory.

Setting the min and max refresh delay to the same value is a nonsensical configuration. Don't do it.

Example HTTP Metadata Provider Configuration

```
<!-- RelyingParty elements above this point -->
<MetadataProvider xsi:type="HTTPMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata"
                  id="MyMetadata"
                  metadataURL="http://example.org/metadata.xml" />
```

Inline Metadata Provider

This Metadata Provider reads metadata from with the provider's configuration element.

It is defined by the element `<MetadataProvider xsi:type="InlineMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attributes:

- **id** - provides a unique, amongst metadata providers, identifier that may be used to reference the provider

and the following optional attributes:

- **failFastInitialization** (added in v2.2) - a boolean flag indicating whether, during startup, the IdP should fail to start if there is a problem fetching metadata (default value: true)
- **requireValidMetadata** (added in v2.2) - a boolean flag that indicates that loaded metadata must be valid, according to the validUntil attributes found in the metadata (default value: true)

The content of the `<MetadataProvider>` element is then either a SAML 2 metadata `<EntitiesDescriptor>` or `<EntityDescriptor>` element.

Example Inline Metadata Provider Configuration

```
<!-- RelyingParty elements above this point -->
<MetadataProvider xsi:type="InlineMetadataProvider" xmlns="urn:mace:shibboleth:2.0:metadata"
                  id="MyInlineMetadata">

   <EntitiesDescriptor xmlns="urn:oasis:names:tc:SAML:2.0:metadata">
      <!-- Rest of SAML 2 metadata -->
   </EntitiesDescriptor>

</MetadataProvider>
```

Define a Metadata Filter (optional)

A metadata provider may also define a metadata filter. This filter operates on incoming metadata and may do things such as validate a metadata's signature, remove unneeded portions of the metadata, add, normalize, or otherwise transform portions of the metadata, etc. The `MetadataFilter` element used to define a filter is always the first element within a metadata provider.

Chaining Metadata Filter

A metadata provider can only have one filter defined within the provider. However, that one filter can be a chaining filter, which in turn can have other filters defined within it. These contained filters are executed in the order they are listed.

The filter is defined by the element `<MetadataFilter xsi:type="ChainingFilter" xmlns="urn:mace:shibboleth:2.0:metadata">`.

The content of the `MetadataFilter` element is other `MetadataFilter` elements.

Example Chaining Metadata Filter

```
<MetadataFilter xsi:type="ChainingFilter" xmlns="urn:mace:shibboleth:2.0:metadata">

    <MetadataFilter xsi:type="SignatureValidation" trustEngineRef="shibboleth.SignatureTrustEngine" />
    <MetadataFilter xsi:type="EntityRoleWhiteList">
        <RetainedRole>samlmd:SPSSODescriptor</RetainedRole>
    </MetadataFilter>

</MetadataFilter>
```

Schema Validation Filter

This metadata filter validates incoming metadata against the SAML metadata schema.

This filter is defined by the element `<MetadataFilter xsi:type="SchemaValidation" xmlns="urn:mace:shibboleth:2.0:metadata">`. It may contain any number of `<ExtensionSchema>` elements whose content is the classpath location for additional XML Schema files to use during validation.

Example Schema Validation Filter

```
<MetadataFilter xsi:type="SchemaValidation" xmlns="urn:mace:shibboleth:2.0:metadata">
   <ExtensionSchema>/schema/foo.xsd</ExtensionSchema>
</MetadataFilter>
```

Required validUntil Filter

This metadata filter requires the presence of the `validUntil` attribute on the root of the retrieved metadata. It may optionally ensure that the value of this attribute is not more than a given amount of time in the future. This ensures that old metadata, which may contain entities which have been removed /revoked, is not used.

This filter is defined by the element `<MetadataFilter xsi:type="RequiredValidUntil" xmlns="urn:mace:shibboleth:2.0:metadata">`.

It may contain the following, optional, attribute:

- **maxValidityInterval** - the interval in ISO8601 duration notation, from now within which the validUntil date must fall. A value of zero indicates no upper limit. Default value: 0

Example Required validUntil Filter

```
<MetadataFilter xsi:type="RequiredValidUntil" xmlns="urn:mace:shibboleth:2.0:metadata"
                maxValidityInterval="P30D" />
```

Signature Validation Filter

This metadata filter validates the signature on the root element of the incoming metadata if the metadata is signed.

The filter is defined by the element `<MetadataFilter xsi:type="SignatureValidation" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following required attribute:

- **trustEngineRef** - A reference to the signature trust engines that will be used to validate the signatures on the metadata

and the optional attribute:

- **requireSignedMetadata** - a boolean flag that requires that incoming metadata be signed (default value: false)

Example Signature Validation Filter

```
<MetadataFilter xsi:type="SignatureValidation" xmlns="urn:mace:shibboleth:2.0:metadata"
                trustEngineRef="shibboleth.MetadataTrustEngine"
                requireSignedMetadata="true" />
```

Most of the trust engines within Shibboleth pull their trust information from SAML metadata. Obviously, you can not use such a trust engine to determine the trustworthiness of the metadata itself, such trust must be bootstrapped. To do this, use of the explicit key static signature trust engine is recommended. The default shibboleth configuration ships with a commented-out, partially configured trust engine, with the id `shibboleth.MetadataTrustEngine`, of this type. If you add additional metadata sources you may simply add more certificates to this trust engine, it is not necessary to define a new trust engine for every trusted CA.

Entity Role WhiteList Filter

Metadata files can grow quite large, especially when they contain the embedded key necessary for XML encryption support. However, only the SP information is usually needed by the IdP, this means that all the IdP metadata being kept around is just taking up memory. This metadata filter removes entity roles, and optional who entity and entities descriptors, that are not needed by the IdP. It is based on a white list because the types of roles that may appear within metadata is not bounded.

This filter is defined by the element `<MetadataFilter xsi:type="EntityRoleWhiteList" xmlns="urn:mace:shibboleth:2.0:metadata">` with the following optional attributes:

- **removeRolelessEntityDescriptors** - a boolean flag that indicates that, after role filtering, an `EntityDescriptor` element should be removed from metadata if it does not contain any roles (default value: true)
- **removeEmptyEntitiesDescriptors** - indicates that, after role and `EntityDescriptor` filtering, an `EntitiesDescriptor` element should be removed from metadata if it does not contain any `EntityDescriptor` or `EntitiesDescriptor` (default value: true)

The contents of the `<MetadataFilter>` element is a list of `<RetainedRole>` elements whose content is the name of the roles that should be retained.

Regardless of the values of the two optional attributes the root element of a metadata document is never removed, even if after filtering it is basically empty.

Example Entity Role WhiteList Filter

```
<MetadataFilter xsi:type="EntityRoleWhiteList" xmlns="urn:mace:shibboleth:2.0:metadata">
    <RetainedRole>samlmd:SPSSODescriptor</RetainedRole>
</MetadataFilter>
```

This example demonstrates filtering out all roles other than the SP SSO descriptor, entity descriptors (because it does not override `removeRolelessEntityDescriptors`) that end up being empty after the role filter, and any entities descriptors that end up being empty (again because we don't override `removeEmptyEntitiesDescriptors`).

Because this filter modifies the metadata, it should always be placed after filters that require the original metadata, for example `SchemaValidation` or `SignatureValidation` filters.

Examples

Additional examples are also available. These provide more complete examples and are contributed by users of the software.

Example 1	Load UK and Swiss federation metadata
Example 2	Refresh InCommon federation metadata every hour