MetadataConfiguration

The Shibboleth IdP generally requires SAML metadata to provision connectivity with relying parties and inform it about their capabilities and technical specifics. While you have the option to operate in a more "promiscuous" way (by enabling profiles for "unverified" RPs), this is relatively rare. In most cases, you will configure metadata sources in order to use the IdP's SAML features; this is done by adding `<MetadataProvider>` elements inside the *metadata-providers.xml* file.

Contents

- Schema names and locations
- MetadataProvider Types
- Attributes
  - Common Attributes
  - Reloading Attributes
  - Dynamic Attributes
  - HTTP Attributes
    - HTTP Connection Attributes
    - HTTP Security Attributes
    - HTTP Proxy Attributes
    - HTTP Caching Attributes
- Child Elements
- Miscellany
  - Multiple Configuration Files
  - Search Ordering
- V2 Compatibility
  - New Capabilities in V3

In a typical (and default) configuration, only one resource is configured for metadata configuration, a file called *metadata-providers.xml* containing a "root" `<MetadataProvider>` element. This could make use of a single resource such as a local file:

Simple file based configuration

```
<MetadataProvider id="LocalMetadata" xsi:type="FilesystemMetadataProvider"
 xmlns="urn:mace:shibboleth:2.0:metadata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:mace:shibboleth:2.0:metadata
http://shibboleth.net/schema/idp/shibboleth-metadata.xsd"
 failFastInitialization="true"
 metadataFile="%{idp.home}/metadata/my-metadata.xml"/>
```

It could also point to a third-party-provided metadata source, usually referred to as a "federation", with filters to verify the signature on the XML document using a pre-established public key:

Loading metadata from a federation metadata server

```
<MetadataProvider id="FederationMetadata"
xsi:type="FileBackedHTTPMetadataProvider"
 xmlns="urn:mace:shibboleth:2.0:metadata"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:mace:shibboleth:2.0:metadata
http://shibboleth.net/schema/idp/shibboleth-metadata.xsd"
 metadataURL="http://federation.example.org/metadata.xml"
 backingFile="%{idp.home}/metadata/federation-metadata.xml">

    <MetadataFilter xsi:type="SignatureValidation" requireSignedRoot="true"
  certificateFile="%{idp.home}/conf/metadata/fed-signing-key.pem"/>

    <MetadataFilter xsi:type="RequiredValidUntil"
maxValidityInterval="P14D"/>
</MetadataProvider>
```

Alternatively, one can use a ChainingMetadataProvider, which combines two or more metadata sources. The *metadata-providers.xml* file that ships with the software declares such a chain "wrapper" by default.

Schema names and locations

Nearly all elements described in this page and its children are defined in the `urn:mace:shibboleth:2.0:metadata` namespace, the schema for which can be located at http://shibboleth.net/schema/idp/shibboleth-metadata.xsd. Throughout this document and its children, this is assumed to be the default XML namespace in effect. The namespace prefix `"metadata:"` is conventionally also bound to this namespace.

The namespace prefix "`security:`" is used to refer to the `urn:mace:shibboleth:2.0:security` namespace, the schema for which can be located at http://shibboleth.net/schema/idp/shibboleth-security.xsd, and is generally used only in advanced scenarios or for compatibility.

The namespace prefix "`resource:`" is used to refer to the `urn:mace:shibboleth:2.0:resource` namespace, the schema for which can be located at http://shibboleth.net/schema/idp/shibboleth-resource.xsd. It is only used by the `ResourceBackedMetadataProvider`

The namespace prefix "`samlmd:`" is used to refer to the `urn:oasis:names:tc:SAML:2.0:metadata` namespace, the schema for which can be located at http://docs.oasis-open.org/security/saml/v2.0/saml-schema-metadata-2.0.xsd

MetadataProvider Types

The precise behavior of any `<MetadataProvider>` element is controlled by the `xsi:type` attribute (see below). The following types are supported and examples are provided for each type. If the `urn:mace:shibboleth:2.0:metadata` namespace is not the default, then a prefix (presumably "metadata:") is required when specifying these types.

xsi:type	Function
ChainingMetadataProvider	A container for an ordered sequence of metadata providers of any type
DynamicHTTPMetadataProvider	A dynamic provider that fetches metadata just-in-time from a suitably configured HTTP server
LocalDynamicMetadataProvider	A dynamic provider that fetches metadata just-in-time from a local source such as a filesystem directory
FilesystemMetadataProvider	A reloading provider that loads (and reloads) a metadata file from the filesystem in a background thread
FileBackedHTTPMetadataProvider	A reloading provider that loads (and reloads) a metadata file from an HTTP server in a background thread
HTTPMetadataProvider	**DEPRECATED**: Use FileBackedHTTPMetadataProvider instead.
ResourceBackedMetadataProvider	A reloading provider that loads (and reloads) a metadata file from a more complex source (such as SVN) i n a background thread

InlineMetadataProvider	A provider that allows metadata to be specified inline

Attributes

Configuration attributes common to two or more metadata providers are listed in the subsections below. Other attributes are specific to the `xsi:type`, and these are documented on the pages specific to each type.

Common Attributes

The following attributes are required on **all** metadata provider types:

Name	Type	Default	Description
`id`	String	required	Identifier for logging, identification for command line reload, etc.
`xsi:type`	String	required	Specifies the exact type of provider to use (from those listed above, or a custom extension type).

The following attributes are common to all metadata provider types except the `ChainingMetadataProvider` type:

Name	Type	Default	Description
`requireValidMetadata`	Boolean	true	Whether candidate metadata found by the resolver must be valid in order to be returned (where validity is implementation specific, but in SAML cases generally depends on a `validUntil` attribute.) If this flag is true, then invalid candidate metadata will not be returned.
`failFastInitialization`	Boolean	true	Whether to fail initialization of the underlying MetadataResolverService (and possibly the IdP as a whole) if the initialization of a metadata provider fails. When false, the IdP may start, and will continue to attempt to reload valid metadata if configured to do so, but operations that require valid metadata will fail until it does.
`sortKey`	Integer		Defines the order in which metadata providers are searched (see below), can only be specified on top level `<MetadataProvider>` elements.
The following are advanced settings supporting a new low-level feature allowing metadata lookup by keys other than the unique entityID and are rarely of use to a deployer.			
`criterionPredicateRegistryRef` 3.3	Bean ID		Identifies the a custom `CriterionPredicateRegistry` bean used in resolving predicates from non-predicate input criteria.
`useDefaultPredicateRegistry` 3.3	Boolean	true	Flag which determines whether the default `CriterionPredicateRegistry` will be used if a custom one is not supplied explicitly.
`satisfyAnyPredicates` 3.3	Boolean	false	Flag which determines whether predicates used in filtering are connected by a logical 'OR' (true) or by logical 'AND' (false).

Reloading Attributes

The following attributes are common to all reloading "batch-oriented" metadata providers (i.e., `FileBackedHTTPMetadataProvider`, `FilesystemMetadataProvider`, and `ResourceBackedMetadataProvider`):

Name	Type	Default	Description
`parserPoolRef`	Bean ID	shibboleth.ParserPool	Identifies a Spring bean for the (OpenSAML) `ParserPool` object used to parse incoming metadata. Generally should not be changed.
`taskTimerRef`	Bean ID		Identifies a Spring bean containing a Java `TaskTimer` used to schedule reloads. When not set, an internal timer is created. Generally should not be changed.
`minRefreshDelay`	Duration	PT30S	Lower bound on the next refresh from the time calculated based on the metadata's expiration.
`maxRefreshDelay`	Duration	PT4H	Upper bound on the next refresh from the time calculated based on the metadata's expiration.
`refreshDelayFactor`	Real Number (strictly between 0.0 and 1.0)	0.75	A factor applied to the initially determined refresh time in order to determine the next refresh time (typically to ensure refresh takes place prior to the metadata's expiration). Attempts to refresh metadata will generally begin around the product of this number and the maximum refresh delay.

`indexesRef` 3.3	Bean ID		Identifies an optional `Set<MetadataIndex>` used to support resolution of metadata based on criteria other than an entityID.
`resolveViaPredicatesOnly` 3.3	Boolean	false	Flag indicating whether resolution may be performed solely by applying predicates to the entire metadata collection, when an entityID input criterion is not supplied.
`expirationWarningThreshold` 3.4	Duration	PT0S (disabled)	For each attempted metadata refresh (whether or not fresh metadata is obtained), if `requireValidMetadata` is true, and there is a `validUntil` XML attribute on the document root element, and the difference between `validUntil` and the current time is less than `expirationWarningThreshold`, the system logs a warning about the impending expiration.

Dynamic Attributes

The following attributes are common to all dynamic metadata providers (i.e., `DynamicHTTPMetadataProvider` and `LocalDynamicMetadataProvider`):

Name	Type	Default	Description
`parserPoolRef`	Bean ID	shibboleth.ParserPool	Identifies a Spring bean for the (OpenSAML) `ParserPool` ject used to parse incoming metadata. Generally should not be changed.
`taskTimerRef`	Bean ID		Identifies a Spring bean containing a Java `TaskTimer` used to schedule reloads. When not set, an internal timer is created. Generally should not be changed.
`refreshDelayFactor`	Real Number (strictly between 0.0 and 1.0)	0.75	A factor applied to the initially determined refresh time in ord to determine the next refresh time (typically to ensure refres takes place prior to the metadata's expiration). Attempts to refresh metadata will generally begin around the product of this number and the maximum refresh delay.
`minCacheDuration`	Duration	PT10M (10 minutes)	The minimum duration for which metadata will be cached before it is refreshed.
`maxCacheDuration`	Duration	PT8H (8 hours)	The maximum duration for which metadata will be cached before it is refreshed.
`maxIdleEntityData`	Duration	PT8H (8 hours)	The maximum duration for which metadata will be allowed t be idle (no requests for it) before it is removed from the cach
`removeIdleEntityData`	Boolean	true	Flag indicating whether idle metadata should be removed.
`cleanupTaskInterval`	Duration	PT30M (30 minutes)	The interval at which the internal cleanup task should run. This task performs background maintenance tasks, such as the removal of expired and idle metadata.
`persistentCacheManagerRef` 3.3	Bean ID		The optional manager for the persistent cache store for resolved metadata. On metadata provider initialization, data present in the persistent cache will be loaded to memory, effectively restoring the state of the provider as closely as possible to that which existed before the previous shutdown Each individual cache entry will only be loaded if 1) the entry is still valid as determined by the internal provider logic, and the entry passes the (optional) predicate supplied via `initi lizationFromCachePredicateRef`.
`persistentCacheManagerDirectory` 3.3	File specification		The directory used for an internally-constructed filesystem-based persistent cache. This is a convenience parameter to avoid specifying a full bean via `persistentC cheManagerRef`. This option will be ignored if `persisten CacheManagerRef` is specified.
`persistentCacheKeyGeneratorRef` 3.3	Bean ID	internal default instance	Identifies a Spring bean for a `Function` which generates th string key used with the cache manager. The default implementation produces the lower-case hex-encoded SHA- digest of the entityID of the `EntityDescriptor`.
`initializeFromPersistentCacheInBackground` 3.3	Boolean	true	Flag indicating whether should initialize from the persistent cache in the background. Initializing from the cache in the background will improve IdP startup times.
`backgroundInitializationFromCacheDelay` 3.3	Duration	PT2S (2 seconds)	The delay after which to schedule the background initializati from the persistent cache when `initializeFromPersist ntCacheInBackground=true`.

`initializationFromCachePredicateRef` 3.3	Bean ID	an "always true" predicate	Identifies a Spring bean for an optional `Predicate` which determines whether a given entity should be loaded from the persistent cache at resolver initialization time.

HTTP Attributes

The following attributes are common to all HTTP metadata providers (i.e., `DynamicHTTPMetadataProvider` and `FileBackedHTTPMetadataProvider`).

An HTTP metadata provider includes a default implementation of the `org.apache.http.client.HttpClient` interface. The attributes in the following subsections control the behavior of the default HTTP client. To override the default client implementation, configure the following attribute:

Name	Type	Default	Description
`httpClientRef`	Bean ID		A reference to an externally defined Spring bean that specifies an `org.apache.http.client.HttpClient` object. This attribute conflicts with and overrides **all** of the HTTP attributes. See the HttpClientConfiguration topic for more information.

Use of the `httpClientRef` attribute precludes the use of any and all of the HTTP attributes in the following subsections.

HTTP Connection Attributes

The following attributes apply to the HTTP connections obtained and managed by an HTTP metadata provider:

Name	Type	Default	Description
`connectionRequestTimeout` 3.3	Duration	Depends on the provider type	The maximum amount of time to wait for a connection to be returned from the HTTP client's connection pool manager. Set to `PT0S` to disable. This attribute is incompatible with `httpClientRef`.
`connectionTimeout` 3.3	Duration	Depends on the provider type	The maximum amount of time to wait to establish a connection with the remote server. Set to `PT0S` to disable. This attribute is incompatible with `httpClientRef`.
`requestTimeout`	Duration	Depends on the provider type	**DEPRECATED**: Use `connectionTimeout` instead.
`socketTimeout` 3.3	Duration	Depends on the provider type	The maximum amount of time to wait between two consecutive packets while reading from the socket connected to the remote server. Set to `PT0S` to disable. This attribute is incompatible with `httpClientRef`.

HTTP Security Attributes

The following security-related attributes apply to any HTTP metadata provider:

Name	Type	Default	Description
`disregardTLSCertificate`	Boolean	false	If true, no TLS certificate checking will take place over an HTTPS connection. This attribute is incompatible with `httpClientRef`. (Be careful with this setting, it is typically only used during testing. See the HttpClientConfiguration topic for more information.)
`disregardSslCertificate`	Boolean	false	**DEPRECATED**: Use `disregardTLSCertificate` instead.
`basicAuthUser`	String		**DEPRECATED**: Use `httpClientSecurityParametersRef` instead.
`basicAuthPassword`	String		**DEPRECATED**: Use `httpClientSecurityParametersRef` instead.
`tlsTrustEngineRef` 3.1	Bean ID		**DEPRECATED**: Use `httpClientSecurityParametersRef` instead.
`httpClientSecurityParametersRef` 3.3	Bean ID		A reference to an externally defined Spring bean that specifies an `org.opensaml.security.httpclient.HttpClientSecurityParameters` instance, which consolidates all HTTP security parameters including advanced TLS usage. This attribute conflicts with and overrides any explicit `TrustEngine` implementation configured as an inline <TLSTrustEngine> element. See the HttpClientConfiguration topic for more information.

HTTP Proxy Attributes

The following attributes configure an HTTP proxy for use with an HTTP metadata provider:

Name	Type	Default	Description
`proxyHost`	String		The hostname of the HTTP proxy through which connections will be made. This attribute is incompatible with `httpClientRef`.
`proxyPort`	String		The port of the HTTP proxy through which connections will be made. This attribute is incompatible with `httpClientRef`.
`proxyUser`	String		The username used with the HTTP proxy through which connections will be made. This attribute is incompatible with `httpClientRef`.
`proxyPassword`	String		The password used with the HTTP proxy through which connections will be made. This attribute is incompatible with `httpClientRef`.

HTTP Caching Attributes

The following attributes configure an HTTP cache on an HTTP metadata provider:

Name	Type	Default	Description
`httpCaching`	`"none"`, `"file"`, or `"memory"`	Depends on the provider type	The type of low-level HTTP caching to perform. There are three choices: • `"none"` indicates the HTTP response is not cached by the client library • `"file"` indicates the HTTP response is written to disk (but will not survive a restart) • `"memory"` indicates the HTTP response is stored in memory This attribute is incompatible with `httpClientRef` and its value may not be specified as a bean property. Some metadata providers, most notably the reloading "batch-oriented" providers, implement HTTP caching at a higher layer and tend to work best with `httpCaching="none"`.
`httpCacheDirectory`	String		If `httpCaching="file"`, this attribute specifies where retrieved files are to be cached. This attribute is incompatible with `httpClientRef`.
`httpMaxCacheEntries`	Integer	`"memory"`: 50 `"file"`: 100	The maximum number of responses written to cache. This attribute is incompatible with `httpClientRef`.
`httpMaxCacheEntrySize`	Integer	`"memory"`: 1048576 (1MB) `"file"`: 10485760 (10MB)	The maximum response body size that may be cached, in bytes. This attribute is incompatible with `httpClientRef`.

Child Elements

The following child elements may be used with all metadata provider types (except the `ChainingMetadataProvider` type):

Name	Cardinality	Description
`<MetadataFilter>`	0 or more	A metadata filter applied to candidate metadata as it flows through the metadata pipeline
`<security:TrustEngine>`	0 or more	**DEPRECATED**: See the note at the bottom of this page

Other allowable child elements are specific to the `xsi:type` of the provider used, and these are documented on the pages specific to each type.

Miscellany

Multiple Configuration Files

As described in the ReloadableServices documentation, the configuration is actually loaded from a bean whose name is specified by the property **idp.service.metadata.resources**, with the default value **shibboleth.MetadataResolverResources** which in turn is defined in services.xml to be a list with one entry: the file *metadata-providers.xml*
You can, if you choose, override this with additional or different files or more advanced sources. Each resource must supply a "top level" `<MetadataProvider>` element with attributes and child elements as described above. Search order amongst multiple top level elements is arbitrated by the `sortKey` attribute, where lower values are processed before higher ones.

Search Ordering

If a specific relying party (as identified by a specific entityID) is duplicated in the metadata sources provided, then which precise entry is chosen is governed by the following rules:

- Metadata sources combined via a chain are searched in the order in which they occur in the chain, and the first entry matching the entityID is returned.
- If multiple "top level" Metadata Providers are provided then they are searched in an order derived from the (numeric) value of the `sortKey` attribute (lowest key first). If no `sortKey` is specified, then the search order is undefined.
- In whatever order of sources is in effect, the first entry matching the entityID is returned.
- If a single metadata source contains multiple entries with the same entityID, then which entry is returned is undefined (exception: invalid entries would be ignored in favor of valid ones in most cases).

V2 Compatibility

A single `<MetadataProvider>` element may be embedded in a legacy *relying-party.xml* file as described in the older documentation. Consult the V2 documentation for this, and do not mix and match this approach with newer configuration features.

During the V2 to V3 upgrade process, the original V2 *relying-party.xml* file is copied to *metadata-providers.xml*, to serve as the metadata configuration for the new version. It's strongly advisable after upgrading to update that file by stripping it of the older content and promote the `<MetadataProvider>` element in it to the root of the file. In the interim all other content in the file except for `<MetadataProvider>` elements (and any referenced `<security:TrustEngine>` elements) is ignored.

The following non-relevant trust engine types often found in a legacy relying-party.xml file are ignored if seen, and are not used for metadata verification (despite the confusing names):

- `Chaining`
- `MetadataExplicitKey`
- `MetadataPKIXX509Credential`
- `MetadataExplicitKeySignature`
- `MetadataPKIXSignature`
- `StaticPKIXX509Credential`

New Capabilities in V3

The V3 metadata configuration syntax is backward-compatible with the V2 `<MetadataProvider>` syntax and adds some useful new shortcuts as well.

> **Avoid deprecated features**
> In anticipation of V4, a number of IdP features have been deprecated in V3. To ensure a seamless upgrade to V4, avoid the use of deprecated features in your V3 deployment. In particular, avoid any metadata-related features deprecated in V3.

You can now provide multiple metadata configuration files (not just multiple metadata sources in one file), as described above.

A `SignatureValidation` filter need not contain a `trustEngineRef` attribute referencing a separately-defined trust engine; instead a certificate file may be specified directly with the `certificateFile` attribute. Alternatively, a PEM-format public key may be supplied inline via the `<PublicKey>` element.

> **Using a TrustEngine element**
> As a child element of the `<MetadataProvider>` element, the use of the `<security:TrustEngine>` element is **DEPRECATED**. If used at all, the element should be declared inside a `SignatureValidation` filter or in most cases simply replaced with the `certificateFile` attribute. See the SignatureValidationFilter topic for more information.