LocalDynamicMetadataProvider

The `LocalDynamicMetadataProvider` fetches metadata from a local source dynamically as needed. The deployer is responsible for populating the local source with data, which may be done while the metadata provider is running. New metadata will be seen automatically the first time it is requested.

> **Use this provider with local metadata**
>
> The `LocalDynamicMetadataProvider` (added in V3.3.0) is used with *local metadata*. See the MetadataManagementBestPractices topic for more information.

Contents

A common use case will be to use a filesystem directory as the local source. A convenience attribute `sourceDirectory` is supplied to facilitate this configuration. In this case, the deployer adds XML files each containing a single `<md:EntityDescriptor>` element to the `sourceDirectory`. By default, when using the `sourceDirectory` the file naming convention will be the lower case hex-encoded SHA-1 digest of the `entityID`, suffixed with ".xml". For example, the metadata for the entity with `entityID` "urn:test:foobar" will be resolved from the `sourceDirectory` with filename "d278c9975472a6b4827b1a8723192b4e99aa969c.xml".

Note that with the default `entityID` digest mechanism, the digested value should not include any leading or trailing whitespace (in particular, no trailing new line character):

Example generating default source key with OpenSSL

```
$ echo -n "urn:test:foobar" | openssl sha1
d278c9975472a6b4827b1a8723192b4e99aa969c
```

Schema Names and location

The `<MetadataProvider>` element and the type `LocalDynamicMetadataProvider` are defined by the `urn:mace:shibboleth:2.0:metadata` schema, which can be located at http://shibboleth.net/schema/idp/shibboleth-metadata.xsd.

Attributes

Any of the Common Attributes or the Dynamic Attributes may be configured. In addition, the following attributes are available on a `LocalDynamicMetadataProvider`:

Attribute	Type	Default	Description
`sourceDirectory`	String		Convenience mechanism for wiring a `FilesystemLoadSaveManager`, loading from the specified source directory in the local filesystem. This attribute will be ignored if `sourceManagerRef` is also specified. Either this attribute or `sourceManagerRef` is required.
`sourceManagerRef`	Bean ID		Identifies the Spring bean for the `XMLObjectLoadSaveManager` which serves as the local source of metadata. Either this attribute or `sourceDirectory` is required.
`sourceKeyGeneratorRef`	Bean ID	internal default instance	Identifies a Spring bean for a `Function` which generates the string key used with the `XMLObjectLoadSaveManager`. The internal default implementation produces the lower-case hex-encoded SHA-1 digest of the entityID of the input criterion. If the manager was effectively specified via `sourceDirectory`, then the internal default implementation suffixes this source key with ".xml".

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).

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 XML parser used to parse 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 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.
`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 to be idle (no requests for it) before it is removed from the cache.
`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 2) the entry passes the (optional) predicate supplied via `initializationFromCachePredicateRef`.
`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 `persistentCacheManagerRef`. This option will be ignored if `persistentCacheManagerRef` is specified.
`persistentCacheKeyGeneratorRef` 3.3	Bean ID	internal default instance	Identifies a Spring bean for a `Function` which generates the string key used with the cache manager. The default implementation produces the lower-case hex-encoded SHA-1 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 initialization from the persistent cache when `initializeFromPersistentCacheInBackground=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.

The source directory and the cache directory must be distinct

The `sourceDirectory` and the `persistentCacheManagerDirectory` (if any) must be distinct. While it is possible, though probably unusual, to enable persistent caching of local metadata, do **NOT** rely on the same directory for both the source and the cache. This would cause the removal of cached metadata to actually remove the underlying metadata from your system.

Child Elements

Any of the following child elements may be specified (in order).

Name	Cardinality	Description
<`MetadataFilter`>	0 or more	A metadata filter applied to candidate metadata as it flows through the metadata pipeline

The <`MetadataFilter`> child element is common to all metadata providers. The `LocalDynamicMetadataProvider` type has no child elements of its own.