AttributeResolverConfiguration

File(s): conf/attribute-resolver.xml, idp.properties

Format: Custom Schema

- Overview
- Reference
  - Properties
  - XML Schema
  - Child Elements
    - <DataConnector>
    - <AttributeDefinition>
    - <PrincipalConnector>
- Attributes
- V2 Compatibility
  - Scripts
  - Null Values
  - Identifying Source Attributes in Dependencies
  - XML Namespaces 3.3
  - New Capabilities for V3
- Deprecated Features
  - Principal Connectors
  - NameID Encoders
  - Transient Identifier Attribute Definitions
  - PrincipalAuthenticationMethod Attribute Definition
  - Dependency Element and sourceAttributeID XML Attribute

Overview

Attribute resolution is about specifying how attributes are to be constructed and then encoded prior to being sent on to a relying party.

You can exercise and debug the behavior of this process using the AACLI tool or the web interface it uses. This is particularly helpful if you're making changes, performing upgrades, etc., to validate the results match in any given case.

In a typical (and default) configuration, only one configuration resource is supplied for controlling attribute resolution, a file called attribute-resolver.xml. All such resources must contain a "root" element named <AttributeResolver> defined in the urn:mace:shibboleth:2.0:resolver XML namespace (exactly the same as in V2).

The example below includes an xsi:schemaLocation attribute that should allow for use of XML-aware editors, but there is no runtime dependency on the web server hosting those schemas.

Simple attribute-resolver.xml file (V3.4)

```xml
<AttributeResolver
  xmlns="urn:mace:shibboleth:2.0:resolver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

  <!-- The EPPN is the "standard" federated username in higher ed. -->
  <!-- For guidelines on the implementation of this attribute, refer -->
  <!-- to the Shibboleth and eduPerson documentation. Above all, do -->
  <!-- not expose a value for this attribute without considering the -->
  <!-- long term implications. -->

  <AttributeDefinition id="eduPersonPrincipalName" xsi:type="Scoped" scope="%{idp.scope}"
    <InputAttributeDefinition ref="uid" />
    <AttributeEncoder xsi:type="SAML1ScopedString" name="urn:mace:dir:attribute-def:eduPersonPrincipalName" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2ScopedString" name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" friendlyName="eduPersonPrincipalName" encodeType="false" />
</AttributeDefinition>
```
The uid is the closest thing to a "standard" LDAP attribute representing a local username, but you should generally *never* expose uid to federated services, as it is rarely globally unique.

```xml
<AttributeDefinition id="uid" xsi:type="PrincipalName">
  <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:uid" encodeType="false" />
  <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0.9.2342.19200300.100.1.1" friendlyName="uid" encodeType="false" />
</AttributeDefinition>
```

In the rest of the world, the email address is the standard identifier, despite the problems with that practice. Consider making the EPPN value the same as your official email addresses whenever possible.

```xml
<AttributeDefinition id="mail" xsi:type="Template">
  <InputAttributeDefinition ref="uid" />
  <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:mail" encodeType="false" />
  <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0.9.2342.19200300.100.1.3" friendlyName="mail" encodeType="false" />
  <Template>
    <![CDATA[
      ${uid}@example.org
    ]]>
  </Template>
  <SourceAttribute>uid</SourceAttribute>
</AttributeDefinition>
```

This is an example of an attribute sourced from a data connector.

```xml
<AttributeDefinition id="eduPersonScopedAffiliation" xsi:type="Scoped" scope="%{idp.scope}"
  <InputDataConnector ref="staticAttributes" attributeNames="affiliation" />
  <AttributeEncoder xsi:type="SAML1ScopedString" name="urn:mace:dir:attribute-def:eduPersonScopedAffiliation" encodeType="false" />
  <AttributeEncoder xsi:type="SAML2ScopedString" name="urn:oid:1.3.6.1.4.1.5923.1.1.1.9" friendlyName="eduPersonScopedAffiliation" encodeType="false" />
</AttributeDefinition>
```

This is an example of an attribute sourced from a data connector.

```xml
<DataConnector id="staticAttributes" xsi:type="Static">
  <Attribute id="affiliation">
    <Value>member</Value>
  </Attribute>
</DataConnector>
```

---

Simple attribute-resolver.xml file (V3.3)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<AttributeResolver
  xmlns="urn:mace:shibboleth:2.0:resolver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <!-- ---------------------------------- -->
  <!-- Attribute Definitions -->
  <!-- ---------------------------------- -->
  <Value>member</Value>
</AttributeResolver>
```

The EPPN is the "standard" federated username in higher ed. For guidelines on the implementation of this attribute, refer...
to the Shibboleth and eduPerson documentation. Above all, do not expose a value for this attribute without considering the long term implications.

```xml
<AttributeDefinition id="eduPersonPrincipalName" xsi:type="Scoped" scope="%{idp.scope}" sourceAttributeID="uid">
    <Dependency ref="uid" />
    <AttributeEncoder xsi:type="SAML1ScopedString" name="urn:mace:dir:attribute-def:eduPersonPrincipalName" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2ScopedString" name="urn:oid:1.3.6.1.4.1.5923.1.1.1.6" friendlyName="eduPersonPrincipalName" encodeType="false" />
</AttributeDefinition>

<!-- The uid is the closest thing to a "standard" LDAP attribute representing a local username, but you should generally *never* expose uid to federated services, as it is rarely globally unique. -->

```xml
<AttributeDefinition id="uid" xsi:type="PrincipalName">
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:uid" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0.9.2342.19200300.100.1.1" friendlyName="uid" encodeType="false" />
</AttributeDefinition>

<!-- In the rest of the world, the email address is the standard identifier, despite the problems with that practice. Consider making the EPPN value the same as your official email addresses whenever possible. -->

```xml
<AttributeDefinition id="mail" xsi:type="Template">
    <Dependency ref="uid" />
    <AttributeEncoder xsi:type="SAML1String" name="urn:mace:dir:attribute-def:mail" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2String" name="urn:oid:0.9.2342.19200300.100.1.3" friendlyName="mail" encodeType="false" />
    <Template>
        <![CDATA[
            $(uid)@example.org
        ]]>
    </Template>
    <SourceAttribute>uid</SourceAttribute>
</AttributeDefinition>

<!-- This is an example of an attribute sourced from a data connector. -->

```xml
<AttributeDefinition id="eduPersonScopedAffiliation" xsi:type="Scoped" scope="%{idp.scope}" sourceAttributeID="affiliation">
    <Dependency ref="staticAttributes" />
    <AttributeEncoder xsi:type="SAML1ScopedString" name="urn:mace:dir:attribute-def:eduPersonScopedAffiliation" encodeType="false" />
    <AttributeEncoder xsi:type="SAML2ScopedString" name="urn:oid:1.3.6.1.4.1.5923.1.1.1.9" friendlyName="eduPersonScopedAffiliation" encodeType="false" />
</AttributeDefinition>

<!-- Data Connectors -->

```xml
<DataConnector id="staticAttributes" xsi:type="Static">
    <Attribute id="affiliation">
        <Value>member</Value>
    </Attribute>
</DataConnector>
</AttributeResolver>
```

Reference

Formally, this is the configuration of the "AttributeResolver" service. By default one file, `attribute-resolver.xml`, defines the attributes to be resolved. Multiple files can be specified by editing the bean referred to by the property `idp.service.attribute.resolver.resources` (default value `shibboleth.AttributeResolverResources` in the `services.xml` file) or changing the property to a different bean name.
Properties

The `attribute-resolver.xml` file can also use property replacement to externalize particular settings such as passwords. By default, these properties are defined in the `idp.properties` file, or you may define your own property files via the `idp.additionalProperties` setting. See the SpringConfiguration topic for more on property syntax.

The properties which affect the resolution process are names starting with `idp.service.attribute.resolver` as described here.

XML Schema

Most elements described in this page and its children are defined in the `urn:mace:shibboleth:2.0:resolver` namespace, the schema for which is located at `http://shibboleth.net/schema/idp/shibboleth-attribute-resolver.xsd`.

Child Elements

<table>
<thead>
<tr>
<th>Name</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DataConnector&gt;</td>
<td>any</td>
<td>Defines connections to sources of data which provide input to attribute definition. These data sources are usually external (databases or attribute stores)</td>
</tr>
<tr>
<td>&lt;AttributeDefinition&gt;</td>
<td>any</td>
<td>Defines the actual attributes and any associated encodings into SAML or other formats</td>
</tr>
<tr>
<td>&lt;PrincipalConnector&gt;</td>
<td>any</td>
<td>Deprecated in V3 and replaced by the mechanisms described under NameIDConsumptionConfiguration. Refer to the V2 documentation for details on the use of this legacy mechanism.</td>
</tr>
</tbody>
</table>

Attributes

The top level `<AttributeResolver>` element has one attribute `id` which is used purely for logging purposes. Child elements can have various attributes; these are described in the appropriate section describing that element.

V2 Compatibility

The overall content and structure is identical to V2. Semantically, the V3 IdP is nearly 100% compatible with V2 attribute configuration. All regressions should be reported via our issue tracker.

Some key exceptions are noted below.

⚠️ The next major version of the IdP will remove this guarantee, and indeed V2 configuration will not work unchanged in V4. It is an aim (but not a guarantee) that any configuration which loads without warning in IdP 3.4 will load in V4. See this page for details.

Scripts

One exception to this compatibility is the Scripted Attribute Definition, where there are inherent limitations supporting more complex scripts, and most existing scripts will be relying on deprecated interfaces. In practice, most existing scripts doing routine things will work unchanged, but more advanced scripts may require modification.

Null Values

In V2, the value collections exposed were literal object types (e.g. String) and null values in an underlying data source were represented as Java nulls (Java collections typically allow null elements). In V3, the values of an `IdPAttribute` are implementations of an interface, `IdPAttributeValue`, that wraps the underlying value. In the case of nulls or empty strings, an explicit type, `EmptyAttributeValue`, is used.

Identifying Source Attributes in Dependencies

The configuration language has a flaw in that the `<AttributeDefinition>` element's `sourceAttributeID` is always optional, but is very ambiguous in different situations. It should have been placed within the `<Dependency>` element but wasn't. As a result, it's either potentially ambiguous if all the dependencies are themselves attribute definitions or explicitly needed if the dependencies are data connectors. Attribute definitions only produce a single attribute (so a dependency on them doesn't need further qualification), while data connectors can produce multiple attributes, with the source ID needed to disambiguate which one is meant as an input.

In V2, it was possible to omit the `sourceAttributeID` attribute when using data connector dependencies. While this may behave consistently if a connector produces only one attribute, it becomes underspecified if the connector is later modified to produce more than one. As of V3.2.0, this is no longer allowed and a warning will be emitted, and the resolution process will fail. You will need to specify an appropriate `sourceAttributeID` to correct the problem.
As of V3.4, all use of `sourceAttributeID` is fully deprecated, and the old `<Dependency>` element itself has been replaced by explicit use of `<InputAttributeDefinition>` and `<InputDataConnector>` elements. Most of the examples have been updated to reflect this syntax.

### XML Namespaces 3.3

In all versions prior to V3.3, the configuration will contain elements in multiple namespaces. With V3.3, every non-deprecated feature can be defined in the main `urn:mace:shibboleth:2.0:resolver` namespace (and thus defaulted to avoid the need for prefixes). The lone exception is the specification of security credentials when connecting to data sources (LDAP mainly).

Apart from the namespaces, nearly all names used have not changed. Hence, for example

```xml
<resolver:AttributeEncoder xsi:type="enc:SAML1ScopedString" name="https://example.org/oldstyle" scopeType="attribute"/>
```

can be rewritten

```xml
<AttributeEncoder xsi:type="SAML1ScopedString" name="https://example.org/oldstyle" scopeType="attribute"/>
```

The two exceptions are the **Scripted Data Connector** and the **Scripted Attribute** definitions, where the name has been extended and so `xsi:type="dc:Script"` should be replaced by `xsi:type="ScriptedDataConnector"` and `xsi:type="ad:Script"` should be replaced by `xsi:type="ScriptedAttribute"`.

### New Capabilities for V3

- **Properties**
- **Activation Conditions on Attribute Definitions, Data Connectors and Attribute Encoders**
- **External Spring Configuration of LDAP, RDBMS, and StoredIdConnector connection details**

### Deprecated Features

A full list of configuration deprecation is available [here](#).

### Principal Connectors

The use of `attribute-resolver.xml` to define external-to-internal subject mapping is deprecated. This function is now independent of the resolver, and is described in `NameIDConsumptionConfiguration`.

By default during upgrades (but not with a fresh install), any `<PrincipalConnector>` elements will be parsed and handled correctly, however deployers are advised to migrate to the newer approach to simplify upgrades to future releases.

### NameID Encoders

The use of `attribute-resolver.xml` to handle encoding attribute data into SAML subject identifiers (i.e., the encoder types `SAMLStringNameIdentifier` and `SAML2StringNameID`) has been deprecated. This function is now independent of the resolver, and is described in `NameIDGenerationConfiguration`.

By default during upgrades (but not with a fresh install), these legacy encoders will be parsed and handled correctly, however deployers are advised to migrate to the newer approach to simplify upgrades to future releases.

### Transient Identifier Attribute Definitions

The use of `attribute-resolver.xml` to define Transient (per-session) identifiers (and eventual encoding by the encoder types `SAMLStringNameIdentifier` and `SAML2StringNameID`) has been deprecated. This function is now independent of the resolver, and is described in `NameIDGenerationConfiguration` and `NameIDConsumptionConfiguration`.

By default, any `<AttributeDefinition>` elements of types `ad:TransientId` or `ad:CryptoTransientId` will be parsed but generally will be superseded by the features described in `NameIDConsumptionConfiguration`, and the older definitions can and should be removed.

### PrincipalAuthenticationMethod Attribute Definition

The `PrincipalAuthenticationMethod` attribute definition is deprecated because the support for managing multiple authentication methods throughout the IdP makes it impractical to expose a single method value.

### Dependency Element and sourceAttributeID XML Attribute
V3.4 has corrected some ambiguities surrounding this element with the new `<InputAttributeDefinition>` and `<InputDataConnector>` elements, and replaces the old `sourceAttributeID` setting. Note that `sourceAttributeID` is ignored once you convert to the newer dependency element syntax, so carefully consider which input data is meant to be consumed by the containing definition.