AttributeRegistryConfiguration

File(s): *conf/attribute-registry.xml, conf/attributes/default-rules.xml, conf/attribute-resolver.xml, conf/attributes/custom/*
Format: Native Spring XML

✓ This is mostly introductory material to explain some concepts. The meat of the documentation you need to operate the system later on starts in the TranscodingRuleConfiguration topic.

Overview

The Attribute Registry service is a new addition to V4 that provides a more advanced way to configure the relationship between the internal IdPAttribute objects that are (for the most part) produced by the Attribute Resolver and the way the data is represented in the protocols supported by the software like SAML, CAS, or in the future OIDC.

In the case of SAML, at least, the actual XML representations and the names used are decoupled from the internal state. The Attribute Registry was designed to address the mapping of data between formats independently of the Attribute Resolver, which is chiefly concerned with how to get data and not how to encode it.

Historically this was addressed with the concept of Attribute Encoders, which were "attached" by the Attribute Resolver to the IdPAttribute objects it produces via Attribute Definitions so the rest of the system could properly encode the XML. This works reasonably well in a lot of cases, and remains supported (though the code under the covers has been completely replaced in this version). We have no plans at present to deprecate this approach.

The purpose of this new registry service is to support additional use cases that were not well, if at all, supported by the old design, such as:

- proxying of attributes obtained from external SSO providers that aren't exactly sourced from the Attribute Resolver
- mapping of SAML (or other protocols) into internal data, such as in the case of `<RequestedAttribute>` elements in SAML metadata
- in many cases, eliminating the need for the SimpleAttributeDefinition plugin if used solely to attach encoders

Using the Attribute Resolver's configuration to express mapping rules gets very awkward or even impossible if there are no scenarios in which the data would ever be produced by it.

There are other advantages, such as the ability to deliver common/reusable mapping rules that have been defaults for a long time, and minimizing clutter in the configuration. We can also better deliver community-supplied translations of display metadata for common attributes for use by the Attribute Consent feature.

Over time, we might expect communities developing their own attribute schemas to provide mapping rules that can be incorporated into the IdP.

General Configuration

The files to load are specified by editing the bean referred to by the property `idp.service.attribute.registry.resources` or changing the property to a different bean name.

The "machinery" for the configuration is spread between a system file and *conf/attribute-registry.xml,* and a directory for custom mapping rules (discussed later) is reserved in **conf/attributes/custom/**

New Installs

New installs load a default set of rules from *conf/attributes/default-rules.xml* to use for a common/standard set of attributes. This contains rules for encoding and decoding of common attributes.

Additionally the configuration loads *conf/attribute-resolver.xml*. The default resolver does not specify any attribute encoders, but referencing the file allows use of encoders for exceptional cases if you prefer to use them.

Upgrades

Upgraded systems with an older *services.xml* file are configured internally to load only the existing AttributeResolverConfiguration resources in order to process the <`AttributeEncoder`> elements within it in order to produce a compatible set of rules to use. The default set of rules supplied with the software is **not** loaded in order to prevent any changes in behavior, including duplication of encoded attributes.

The default resource set after upgrades is meant as a compatibility baseline. Once a determination on the desired approach to configure behavior in the future is made, upgraded systems can be adjusted to add in a **shibboleth.AttributeRegistryResources** bean and an explicit decision made as to what resources to load by editing *services.xml* to do so.

Another note on upgrades is that in the unlikely event that you have any attribute display name or description metadata present for an attribute definition but do **not** have an encoder, the compatibility support is **not** going to attach that display metadata to the resulting attribute. This does not seem like a likely scenario, but it's a change.

Transcoding Rules

The mapping rules that make up the registry are referred to as "transcoding rules" because they support both encoding and decoding attributes to and from e.g., SAML, thus the plugins that support this are called AttributeTranscoders. The mappings are turned into objects of the class TranscodingRule, which are indexed in a variety of ways to enable the system to find the right rules to map to and from the various supported objects.

The system is designed with a high degree of flexibility for defining and loading the mapping rules, but the primary mechanism defined for now is based on properties or maps keyed by some pre-defined strings. Most often the map values will also be strings, which allows use of a Java property file as as an alternative way of defining a rule without using XML.

A suggested pattern is to use rules in XML for supplying widely used defaults and standard conventions "in bulk", but use individual property files to add custom one-off or local rules to supplement the defaults. While the whole system is subject to deployer change, using defaults where possible makes configurations more consistent and avoids needless local divergence (i.e., do you really need to call e-mail address something other than "mail" internally?).

This approach is implemented in the delivered software by adding an **attributes** directory containing the *default-rules.xml* file, a set of (obviously) default rules built using maps, and also containing a **custom** directory to store property files containing custom/local mapping rules.

For detailed documentation on how to create and understand the rules themselves, refer to the TranscodingRuleConfiguration topic.

Reference

Properties

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

Beans

The service resource beans are mentioned above and included in the reference here. The rest are noted in the TranscodingRuleConfiguration topic.