ResolverScriptAttributeDefinition

Script Attribute Definition

The script attribute definition allows attributes to be constructed by the execution of a JSR-223 supported script. Currently only ECMAScript (previously known as Javascript) is supported in a default IdP distribution. Others may be added, however.

> If your IdP is clustered using Terracotta, you may experience out of memory conditions due to the Javascript optimizer creating lots of temporary classes. You can work around this by replacing lib/rhino-V###.jar in the IdP distribution with a copy of the .jar with the org/mozilla /javascript/optimizer directory removed.

1. Define the Definition

The definition is defined with the element `<resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad">` with the following required attribute:

- **id** - assigns a unique, within the resolver, identifier that may be used to reference this definition

and the following optional attribute:

- **language** - the JSR-223 name of the scripting language used within the script (default value: javascript)
- **dependencyOnly** - a boolean flag that indicates the attribute produced by this definition is used only by other resolver components are should not be released from the resolver (default value: false)

Script Attribute Definition with inline script

```
<resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
                              id="UNIQUE_ID">

     <!-- Remaining configuration from the next step go here -->

</resolver:AttributeDefinition>
```

2. Define Dependencies

It is very common for one component, like attribute definitions, within the attribute resolver to depend on information retrieved or constructed from another component.

Dependencies are expressed by the `<resolver:Dependency>` with a `ref` attribute whose value is the unique ID of the attribute definition or the data connector that this connector depends on.

Script Attribute Definition with Dependencies

```
<resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
                              id="UNIQUE_ID">

     <resolver:Dependency ref="DEFINITION_ID_1" />
     <resolver:Dependency ref="DEFINITION_ID_2" />
     <resolver:Dependency ref="CONNECTOR_ID_3" />
     <resolver:Dependency ref="CONNECTOR_ID_4" />

     <!-- Remaining configuration from the next step go here -->

</resolver:AttributeDefinition>
```

3. Define the Script

The script to be executed may be referenced in one of two ways, both represented as an element contained in the `<resolver:AttributeDefinition>` element. The `<Script>` element allows a script to defined inline, that is the content of the element is the script itself. Alternatively the `<ScriptFile>` element defines the location, on the filesystem, of a file containing the script to be executed.

Attribute IDs may contain illegal characters

Some scripting languages place certain restrictions on variable names (e.g. may not contain a "-"). Ensure that any attribute you wish to use does not contain such an illegal character in its ID. If it does use a Simple Attribute Definition to create a new attribute, with a different ID, whose source attribute is the attribute with the problematic ID.

Script Attribute Definition with Inline Script

```
<resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
                              id="fullName">

     <!-- Dependency information would go here -->

    <Script><![CDATA[
        importPackage(Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);

        fullname = new BasicAttribute("fullname");
        fullname.getValues().add(givenName.getValues().get(0) + " " + sn.getValues().get(0));
    ]]></Script>
</resolver:AttributeDefinition>
```

Script Attribute Definition with Script File

```
<resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
                              id="UNIQUE_ID" >

    <!-- Dependency information would go here -->

    <ScriptFile>/usr/local/shibboleth-idp/script/myScript.js</ScriptFile>

</resolver:AttributeDefinition>
```

Information Available to the Script

The following information is made available to an executing script:

- A variable whose name is the ID of the attribute definition. An instance of *edu.internet2.middleware.shibboleth.common.attribute.BaseAttribute* must be created, populated, and assigned to this variable in order to expose the result of this script.
- A variable named **requestContext** containing the *edu.internet2.middleware.shibboleth.common.profile.provider.BaseSAMLProfileRequestContext* for the current resolution request. The object represented by that variable contains data as indicated by all the following interfaces:
  - BaseSAMLProfileRequestContext
  - ProfileRequestContext
  - AttributeRequestContext
  - SAMLMessageContext
  - MessageContext
- A variable for each attribute produced by the defined dependencies of this definition (the attribute itself if the dependency is an attribute definition, every attribute resolved by the data connector if the dependency is a dataconnector). The variable's name will be that of the ID of the attribute from the dependency. In the event that more than one dependency produces attributes with the same ID the values of all of those attributes will be merged and made available to the script. Note that the `sourceAttribute` attribute of the configuration is not used.

Do not change the contents of input attributes

Except in the specific case noted in IdPJava1.8 attribute should **never** be modified by scripts since this will yield undefined results.

Logging Within a Script

The same logging framework used throughout the IdP (SLF4J) may also be used for logging within a script. First import the package `org.slf4j` and then obtain an `org.slf4j.Logger` object from an `org.slf4j.LoggerFactory`. The logging category name used is arbitrary, but must be consistent with the IdP's logging configuration. Logging levels available are: error, warn, info, debug, trace.
The string passed to the `LoggerFactory.getLogger` method should be the name of an existing logger element, defined in logging.xml.

For more information on configuring logging within the IdP, see the IdP Logging topic.

```
 <resolver:AttributeDefinition xsi:type="Script" xmlns="urn:mace:shibboleth:2.0:resolver:ad"
                              id="scriptTest">
    <Script><![CDATA[
        importPackage(Packages.edu.internet2.middleware.shibboleth.common.attribute.provider);
        importPackage(Packages.org.slf4j);

        logger = LoggerFactory.getLogger("edu.internet2.middleware.shibboleth.resolver.Script.scriptTest");

        scriptTest = new BasicAttribute("scriptTest");
        scriptTest.getValues().add("foo");
        scriptTest.getValues().add("bar");

        logger.info("Values of scriptTest were: " + scriptTest.getValues());
    ]]></Script>
</resolver:AttributeDefinition>
```

Java8

Java 8 introduced an incompatible scripting language. In nearly all cases updating to java 8 will require configuration changes. This is discussed here.

Examples

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

Example 1	Generates an opaque identifier from an attribute.
Example 2	Generates eduPersonAffiliation based on group membership.
Example 3	Adds common-lib-terms to eduPersonEntitlement based on the user's affiliation.