ConditionScript

The `<ConditionScript>` element contains a script (or a reference to a script) that ultimately applies an implementation of `Predicate<EntityDescriptor>` to a given entity descriptor.

Software version requirement

This feature requires IdP V3.4 or later.

The `<ConditionScript>` element implicitly iterates over all entity descriptors in the metadata pipeline. For each entity descriptor, the parent `<MetadataFilter>` element acts on the input entity descriptor if (and only if) the predicate evaluates to true. The action taken depends on the type of metadata filter.

The `<ConditionScript>` may be a child of the following filters:

- PredicateMetadataFilter
- NameIDFormatFilter
- EntityAttributesFilter

Schema

The `<ConditionScript>` element is a configuration element of type `ScriptType`. Both the element and its type are defined by the `urn:mace:shibboleth:2.0:metadata` schema, which can be located at http://shibboleth.net/schema/idp/shibboleth-metadata.xsd.

The following sections describe the attributes and elements of the `ScriptType` type.

Attributes

An element of type `ScriptType` has the following XML attributes:

Name	Type	Use	Default	Description
`language`	string	optional	`"javascript"`	Defines the JSR-223 language to use. The default is ECMAScript using either the Rhino (Java 7) or Nashorn (Java 8) engines.
`customObjectRef` 3.2	string	optional		The ID of a Spring bean defined elsewhere in the configuration.

If the `customObjectRef` attribute is present, the result of the referenced Spring bean is made available to the script in a variable named `custom`. This is in addition to the normal script context discussed below.

Child Elements

An element of type `ScriptType` has the following child elements:

Name	Cardinality	Description
`<Script>`	Exactly One	An inline script
`<ScriptFile>`		Path to a local file or classpath resource containing the script

The script may be stored in a local file (with `<ScriptFile>`) or written inline (with `<Script>`). An inline script should be wrapped with a CDATA section to prevent interpretation of any special XML characters that may be included in the script.

Always wrap inline scripts with a CDATA section

Always wrap inline scripts with a CDATA section, even if the script contains no special XML characters. This will future-proof your script.

Script Context

A script contained by a `<ConditionScript>` element has access to an object called `input` by convention. The actual `input` argument is an instance of a class that implements the `EntityDescriptor` interface. Additionally the script has access to an object called `custom`. This is the bean specified using the `customObjectRef` attribute, if present, and null if not..

Examples

The following trivial implementation of `Predicate<EntityDescriptor>` always returns false regardless of the `input` argument:

A trivial implementation of Predicate<EntityDescriptor>

```
<ConditionScript>
    <Script>
    <![CDATA[
        "use strict";
                false;
    ]]>
    </Script>
</ConditionScript>
```

A more complex example might use the `custom` object to help in the definition

A trivial implementation of Function<T, Predicate<EntityDescriptor>>

```
<ConditionScript customObjectRef="BeanID">
    <Script>
    <![CDATA[
        "use strict";
        var someCondition = function(entityID) {
            // Good stuff
        }

        var result;
        // CustomObjectRef points to a <util:map> where the key is a string and the value is an 'interesting
bean'
                if (someCondition(input.getEntityID())) {
            result = custom["myFirstBean"].someFunction(input);
        } else {
            result = custom["mySecondBean"].someOtherFunction(input);
        }
        result;
    ]]>
    </Script>
</ConditionScript>
```

Note that both formal parameter names (`t` and `entity`) are arbitrary. A nontrivial script would presumably substitute a more meaningful name for the formal parameter `t`.