

Programming Guide for SAML-XACML Extension Library and XACML object providers

Introduction

This document contains a introduction on how to work with the SAML-XACML extension to OpenSAML and XACML object providers, which is now a part of the OpenSAML codebase.

Will first go through the basic around the extension and object providers structure of package and such. Then programming examples will be provided along with explanation.

Pre-requisites

All the needed jar files for running the test suite is located in the lib directory, so just add the jars to the classpath.

Packages

Short introduction to the packages is this library and some configurations issues.

org.opensaml.xacml.profile.saml This package contains all the interfaces that is used for the SAML-XACML profile. It defines the XML elements, their type, namespaces and prefixes.

org.opensaml.xacml.profile.saml.impl All interfaces or XML elements has four classes associated with them, these are

1. *Impl, is the actual implementation
2. *Builder, where the object is created
3. *Marshaller, turns the Java object into DOM representation
4. *Unmarshaller, gets a DOM object and outputs the correct Java object

org.opensaml.xacml.ctx Contains all the object providers for XACML which has XML elements in the XACML-context namespace (urn:oasis:names:tc:xacml:2.0:context:schema:os).

org.opensaml.xacml.policy Contains all the object providers for XACML which has XML elements in the XACML-context namespace (urn:oasis:names:tc:xacml:2.0:policy:schema:os).

Creation and use of XACML-SAML profile

We will in this section go through how to create two of the main elements of this extension, `XACMLAuthzDecisionQueryType` and `XACMLAuthzDecisionStatementType`.

Creation of `XACMLAuthzDecisionQueryType`

We'll begin with `XACMLAuthzDecisionQueryType`, which is used to request authorization decision. First we have to get an instance of the `XMLObjectBuilderFactory` which is similar for both examples.

```
//Getting the build factory
XMLObjectBuilderFactory builderFactory =
org.opensaml.xml.Configuration.getBuilderFactory();
```

`XACMLAuthzDecisionQueryType` is an extension of `RequestAbstractType` and it contains a `RequestType` object. The creation and use is very simple

1. Get the right builder for this element. As input we have `XACMLAuthzDecisionQueryType.DEFAULT_ELEMENT_NAME_XACML20`, this is the fully qualifying name of the element when we use XACML 2.0. If one want to use another version the number switch to either 10, 1.1 or 30.
2. Then build the object. The *buildObject* method also needs input on what XACML version we attend to use. `SAMLProfileConstants.SAML20XACML20P_NS` is the namespace for version of XACML we want to use in the SAMLXACML protocol and `SAMLProfileConstants.SAML20XACMLPROTOCOL_PREFIX` is the prefix for this element.
3. Set the needed elements

The code snippet below illustrates the points above, the numbers in the comments are corresponding with the above.

```
//1
XACMLAuthzDecisionQueryTypeImplBuilder xacmlDecisionQueryBuilder =
    (XACMLAuthzDecisionQueryTypeImplBuilder)
    builderFactory.getBuilder(XACMLAuthzDecisionQueryType.DEFAULT_ELEMENT_NAME_XACML20);

//2
XACMLAuthzDecisionQueryType xacmlQuery = xacmlDecisionQueryBuilder.buildObject(
    SAMLProfileConstants.SAML20XACML20P_NS,
    XACMLAuthzDecisionQueryType.DEFAULT_ELEMENT_LOCAL_NAME,
    SAMLProfileConstants.SAML20XACMLPROTOCOL_PREFIX);

//Set the needed elements
xacmlQuery.setID("1234");
xacmlQuery.setDestination("localhost");
xacmlQuery.setIssuer(objectissuer);
xacmlQuery.setVersion(org.opensaml.common.SAMLVersion.VERSION_20);
xacmlQuery.setRequest(objectxacmlrequest);
```

```
xacmlQuery.setIssueInstant(issueInstantQ);
```

In the last code section the request is set. The request is just input from a file first we write the XML file to a DOM object, then we unmarshall the DOM object into a `RequestType` object, which we then set in the `XACMLAuthzDecisionQueryType`.

```
InputStream in = new FileInputStream(file);
BasicParserPool pool = new BasicParserPool();
Document documentXACMLRequest = pool.parse(in);
```

```
UnmarshallerFactory marshallerFactory =
    org.opensaml.xml.Configuration.getUnmarshallerFactory();
```

```
Unmarshaller requestUnmarshaller =
    marshallerFactory.getUnmarshaller(RequestType.DEFAULT_ELEMENT_NAME);
```

```
RequestType request = (RequestType)requestUnmarshaller.
    unmarshall(documentXACMLRequest.getDocumentElement());
```

The result should look something like this

```
<xacml-samlp:XACMLAuthzDecisionQuery
  xmlns:xacml-samlp="urn:oasis:names:tc:xacml:2.0:profile:saml2.0:v2:schema:protocol"
  Destination="localhost" ID="1234" InputContextOnly="false"
  IssueInstant="2007-11-13T13:52:14.232Z" ReturnContext="false" Version="2.0">
  <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
    CN=Hakon Sagehaug,OU=bccs.uib.no,O=NorduGrid,O=Grid
  </saml:Issuer>
  <Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
    <Subject>...</Subject>
    <Resource>...</Resource>
    <Action>...</Action>
    <Environment>...</Environment>
  </Request>
</xacml-samlp:XACMLAuthzDecisionQuery>
```

One can also create the `RequestType` with the use of the XACML providers which is also made for this extension.

Creation of `XACMLAuthzDecisionStatementType`

The reason why the use of `XACMLAuthzDecisionStatementType` is shown, is that it's an important difference from `XACMLAuthzDecisionQueryType`. While `XACMLAuthzDecisionQueryType` creates a new XML element, `XACMLAuthzDecisionStatementType` just creates a new type of `Statement`. In the SAML specification, lines 3133-3137, these types SHOULD be included in SAML instances by means of an `xsi:type` attribute.

```
<Assertion ...>
  ..
  <Statement xsi:type="xacml-saml:XACMLAuthzDecisionStatementType"
```

```

        xmlns:xacml-saml=
            "urn:oasis:names:tc:xacml:2.0:profile:saml2.0:v2:schema:assertion">
        <Response...>...</Response>
        <Request...>...</Request>
    </Statement>
    ..
</Assertion>

```

An other difference is that Statement is contained inside an Assertion element. XACMLAuthzDecisionStatementType can contain a RequestType and/or ResponseType.

First we have to get a builder for the Assertion and the XACMLAuthzDecisionStatementType.

```

XMLObjectBuilderFactory builderFactory =
    org.opensaml.xml.Configuration.getBuilderFactory();

AssertionBuilder objectassertion = (AssertionBuilder)
    builderFactory.getBuilder(Assertion.DEFAULT_ELEMENT_NAME);

XACMLAuthzDecisionStatementTypeImplBuilder xacmlauthz =
    (XACMLAuthzDecisionStatementTypeImplBuilder)builderFactory.
        getBuilder(XACMLAuthzDecisionStatementType.TYPE_NAME_XACML20);

```

When we want to build a XACMLAuthzDecisionStatementType we just need to call *buildObject* as before. Here is the code

```

XACMLAuthzDecisionStatementType objectxacmlauthz = xacmlauthz.buildObject(
    Statement.DEFAULT_ELEMENT_NAME,
    XACMLAuthzDecisionStatementType.TYPE_NAME_XACML20);

```

We can then set the request or response as we did above. To add the object into the Assertion we do like this

```

objectassertion.getStatements().add(objectxacmlauthz);

```

To retrieve the XACMLAuthzDecisionStatementType we do like this

```

List<Statement> statements = objectassertion.
    getStatements(XACMLAuthzDecisionStatement.TYPE_NAME_XACML20);

```

```

XACMLAuthzDecisionStatement xacmlS =
    (XACMLAuthzDecisionStatement)statements.get(0);

```

where objectassertion is the Assertion object. If everything went well the result should look similar to this

```

<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="9812" IssueInstant="2007-09-22T22:00:00.000Z" Version="2.0">
    <saml:Issuer SPProvidedID="dvsvdsv">
        https://XACMLPDP.example.com
    </saml:Issuer>
    <saml:Subject></saml:Subject>
    <saml:Statement

```

```

xmlns:xacml-saml=
  "urn:oasis:names:tc:xacml:2.0:profile:saml2.0:v2:schema:assertion"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xacml-saml:XACMLAuthzDecisionStatementType">
<Request>
  <Subject>...</Subject>
  <Resource>...</Resource>
  <Action>...</Action>
</Request>
<Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <Result
    ResourceId="CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-dteam/dteam">
    <Decision>Permit</Decision>
    <Status>...</Status>
    <Obligations>
      <Obligation FulfillOn="Permit" ObligationId="MappingData">
        <AttributeAssignment AttributeId="User"
          DataType="http://www.w3.org/2001/XMLSchema#string">
          .poolname
        </AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
</Response>
</saml:Statement>
</saml:Assertion>

```

Creation and use of XACML object providers

Making a XACML Requests

Making a XACML Request involves a few steps these are in general

- Creating Subject, Action, Environment and Resource elements
- Attribute elements which holds all the meta information about the attribute
- AttributeValue that holds the actual value of the attribute
- Adding the attribute to a Subject, Action, Environment or a Resource element
- Then adding thees to a XACML Request

This section will go through how to create a subject and adding it to the XACML Request, this example is gathered from the class **no.bccs.parallab.sample.SampleProgramXACMLProvider** and the method *buildXACMLRequest*.

The input for the method is a map of the subject values and their id like this

```

Map<String,String> subjectMap = new HashMap<String,String>();
subjectMap.put
("http://authz-interop.org/xacml/2.0/subject/username", "username");
subjectMap.put("http://authz-interop.org/xacml/2.0/subject/subject-x509-id",
"/O=Grid/O=myowngrid/OU=bccs.uib.no/CN=Hakon Sagehaug");

```

Then we need to get a builder for the AttributeType and XSString, which will hold the actual value of the attribute

```

AttributeTypeImplBuilder attributeBuilder = (AttributeTypeImplBuilder)
builderFactory.getBuilder(AttributeType.DEFAULT_ELEMENT_NAME);

```

```

StringBuilder xmlStringBuilder = (XSStringBuilder)
builderFactory.getBuilder(XSString.TYPE_NAME);

```

Now we need to iterate over the set of keys in the Map of subject values. After that we get out the object with the key we extracted from the key set. This values is used to set the value of the XSString object. Then we create a AttributeType and set some meta information and finally the value of the attribute. In the end we need to add the newly created AttributeValue to the subject, this is the final line in the example.

```

for(String subjectKey : subjectMap.keySet()){
    String subjectString = subjectMap.get(subjectKey);

    XSString subjectValue = xmlStringBuilder.buildObject(
        AttributeValueType.DEFAULT_ELEMENT_NAME,
        XSString.TYPE_NAME);
    //setting the value of the attribute value element
    subjectValue.setValue(subjectString);

    AttributeType xacmlSubjectAttribute = attributeBuilder.buildObject(
        XACMLConstants.XACML2OCTX_NS,
        AttributeType.DEFAULT_ELEMENT_LOCAL_NAME,
        XACMLConstants.XACMLCONTEXT_PREFIX);

    xacmlSubjectAttribute.setIssuer("issuer");
    xacmlSubjectAttribute.setAttributeID(subjectKey);
    xacmlSubjectAttribute.getAttributeValues().add(subjectValue);

    subject.getAttributes().add(xacmlSubjectAttribute);
}

```

Now the subject has been created we can add to the XACML Request. Like shown below we first get a builder for the Request, then we add the subject to it

```

RequestTypeImplBuilder requestBuilder = (RequestTypeImplBuilder)
builderFactory.getBuilder(RequestType.DEFAULT_ELEMENT_NAME);

```

```

RequestType request = requestBuilder.buildObject();

```

```
request.getSubjects().add(subject);
```

When all this is done we are left with a XACML Request containing a Subject elements that has one Attribute with two values like shown below

```
<?xml version="1.0" encoding="UTF-8"?>
<xacml-context:Request
  xmlns:xacml-context="urn:oasis:names:tc:xacml:2.0:context:schema:os">
  <xacml-context:Subject>
    <xacml-context:Attribute
      AttributeId="http://authz-interop.org/xacml/2.0/subject/username">
      <xacml-context:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">
        username
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
    <xacml-context:Attribute
      AttributeId="http://authz-interop.org/xacml/2.0/subject/subject-x509-id">
      <xacml-context:AttributeValue xmlns:xs="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xs:string">
        /O=Grid/O=myowngrid/OU=bccs.uib.no/CN=Hakon Sagehaug
      </xacml-context:AttributeValue>
    </xacml-context:Attribute>
  </xacml-context:Subject>
</xacml-context:Request>
```

Of course one have to add the action, resource and environment to the request but this is all done in a similar fashion.

Unmarshalling a SAML Response for extracting Obligations values

I'll here show; if you start with a DOM element how you get it into Java objects and work with them. I'll start with a SAML Response message from a file, but this could also just arrived over the wire, as long as it is a DOM object. Here is how the file looks

```
<?xml version="1.0"?>
<Response xmlns="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  ID="_535e201c-fe92-4a6b-8a72-9837eb12ecf5" Version="2.0"
  InResponseTo="_3583b0f7-ba9f-48e4-90f6-478060dd131a"
  IssueInstant="2007-11-28T18:09:44.119+01:00">
  <Issuerxmlns="urn:oasis:names:tc:SAML:2.0:assertion">
    CN=cert-pbox-02.cnaf.infn.it,L=CNAF,OU=Host,O=INFN,C=IT
  </Issuer>
  <Status>
    <StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </Status>
```

```

<Assertion xmlns="urn:oasis:names:tc:SAML:2.0:assertion"
  ID="_0a98e91a-1298-4997-86d5-b9284636c44e" Version="2.0"
  IssueInstant="2007-11-28T18:09:44.121+01:00">
  <Issuer>CN=cert-pbox-02.cnaf.infn.it,L=CNAF,OU=Host,O=INFN,C=IT</Issuer>
  <Statement
    xmlns:xacml-saml=
      "urn:oasis:names:tc:xacml:2.0:profile:saml2.0:v2:schema:assertion"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:type="xacml-saml:XACMLAuthzDecisionStatementType">
  <Response xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
    <Result ResourceId="CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-dteam/dteam">
      <Decision>Permit</Decision>
      <Status>
        <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
      </Status>
      <Obligations xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
        <Obligation FulfillOn="Permit" ObligationId="MappingData">
          <AttributeAssignment AttributeId="UID"
            DataType="http://www.w3.org/2001/XMLSchema#string">
            001
          </AttributeAssignment>
        </Obligation>
        <Obligation FulfillOn="Permit" ObligationId="MappingData">
          <AttributeAssignment AttributeId="GID"
            DataType="http://www.w3.org/2001/XMLSchema#string">
            005
          </AttributeAssignment>
        </Obligation>
      </Obligations>
      <Obligations xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
        <Obligation FulfillOn="Permit" ObligationId="LogAccess">
          <AttributeAssignment AttributeId="log"
            DataType="http://www.w3.org/2001/XMLSchema#string">
            Logging is good
          </AttributeAssignment>
        </Obligation>
      </Obligations>
    </Result>
  </Response>
</Statement>
</Assertion>
</Response>

```

The main goal in the end is to get out information about the obligation inside this XML file.

First we get the XML file into a DOM object like shown below.

```

InputStream in = new FileInputStream(file);
BasicParserPool pool = new BasicParserPool();
Document doc = pool.parse(in);

```

After this we first get the unmarshaller for a SAML response element, then we unmarshall the DOM object into a Response object.

```
// Get the unmarshaller factory
UnmarshallerFactory unMarshallerFactory =
org.opensaml.xml.Configuration.getUnmarshallerFactory();

Unmarshaller queryUnmarshaller =
unMarshallerFactory.getUnmarshaller(Response.DEFAULT_ELEMENT_NAME);

Response response = (Response)queryUnmarshaller.
unmarshall(doc.getDocumentElement());
```

Once we've unmarshalled the XML file, we can get the elements by calling get methods on the Response object. The listing below shows how to get the assertion and statements of type XACMLAuthzDecisionStatementType from a SAMLResponse.

```
Assertion assertion = response.getAssertions().get(0);
List<Statement> statements = assertion.
getStatements(XACMLAuthzDecisionStatementType.TYPE_NAME_XACML20);
```

I use a 0(zero), for getting the first assertion. Then we can get the first XACMLAuthzDecisionStatementType, so the ResponseType,ResultType.

```
XACMLAuthzDecisionStatementType xacmlS =
(XACMLAuthzDecisionStatementType)statements.get(0);

ResponseType xacmlResponse = xacmlS.getResponse();
```

```
ResultType result = xacmlResponse.getResult();
```

When we have the result we can get a list of obligations elements. We can have multiple obligations elements and multiple obligation inside a obligations, like this

```
<Obligations xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
  <Obligation FulfillOn="Permit" ObligationId="MappingData">
    <AttributeAssignment AttributeId="UID"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      001
    </AttributeAssignment>
  </Obligation>
  <Obligation FulfillOn="Permit" ObligationId="MappingData">
    <AttributeAssignment AttributeId="GID"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      005
    </AttributeAssignment>
  </Obligation>
</Obligations>
```

First we have to get the obligations from the result like this

```
ObligationsType obligations = result.getObligations();
```

```
List<ObligationType> obligationList = obligations.getObligations();
```

Then we iterate over all the obligations in a double for loop.

```
for(ObligationType oblig:obligationList){
    List<AttributeAssignmentType> attributeAssignmentsList =
        oblig.getAttributeAssignments();
    for(AttributeAssignmentType attributeAssignment :attributeAssignmentsList)
    {
        log.debug("Id of the obligations is {} and the value is {}",
            attributeAssignment.getAttributeId(),
            attributeAssignment.getValue()
        );
    }
}
```

And the output is like this

```
UID 001
GID 005
```

Unmarshall a Environment element

This section provides a example on how to unmarshall a XACML Environment element, to extract values contained inside AttributeValueType element. We start with the Environment XML file like this one

```
<xacml-context:Environment xmlns:xacml-context=
    "urn:oasis:names:tc:xacml:2.0:context:schema:os"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oasis:names:tc:xacml:1.0:context
    cs-xacml-schema-context-01.xsd">
<xacml-context:Attribute AttributeId="urn:oblig:id"
    DataType="http://www.w3.org/2001/XMLSchema#anyURI">
<xacml-context:AttributeValue>
    urn:globus:local-user-name:obj
</xacml-context:AttributeValue>
<xacml-context:AttributeValue>
    urn:globus:local-group-name:obj
</xacml-context:AttributeValue>
</xacml-context:Attribute>
</xacml-context:Environment>
```

First we parse the XML file into a DOM object then we unmarshall the DOM object into a EnvironmentType Java object.

```
InputStream in = new FileInputStream(file);
BasicParserPool pool = new BasicParserPool();
Document doc = pool.parse(in);
```

```
UnmarshallerFactory unMarshallerFactory =
    org.opensaml.xml.Configuration.getUnmarshallerFactory();
```

```
Unmarshaller environUnmarshaller =
unMarshallerFactory.getUnmarshaller(EnvironmentType.DEFAULT_ELEMENT_NAME);
```

```
EnvironmentType xacmlEnvironment = (EnvironmentType)
environUnmarshaller.unmarshall(doc.getDocumentElement());
```

After this we use the *getAttributes()* method for getting all the attribute values inside the Environment. Then we iterate over the values and extract the value.

```
List<AttributeType> attributes = xacmlEnvironment.getAttributes();

for(AttributeType attribute : attributes){
    list<XMLObject> attributeValues = attribute.getAttributeValues();
    log.debug("Id or the attribute:{}",attribute.getAttributeID());
    for(XMLObject value : attributeValues){
        AttributeValueType attributeValue = (AttributeValueType) value;
        log.debug("Value of the attribute:{}",attributeValue.getValue());
    }
}
```

The output should be this

```
Id or the attribute:urn:oblig:id
Value of the attribute:urn:globus:local-user-name:obj
Value of the attribute:urn:globus:local-group-name:obj
```

Make use of the obligation handling code in org.opensaml.xacml.provider package

The **org.opensaml.xacml.provider** package provides code for handling obligations inside a XACML Result, that are returned from a Policy Decision Point(PDP). There are three main classes in package, these are

- **BaseObligationHandler**, base class for handlers, this is an abstract class, which has one method *-evaluateObligation()* that needs to be implemented for each obligation that needs to be handled.
- **ObligationProcessingContext**, the context in which the obligation is to be processed. This class contains the XACML Result, which holds the information about the obligation.
- **ObligationService**, this is where the execution takes place. It has a list of obligation handlers and iterates over them and executes *evaluateObligation()* on each of them.

Print obligation service

The test scenario is just to print out the values in the obligation element, but makes use of the classes described above. So then I make a new class named

PrintObligation that extends **BaseObligationHandler** and implements *evaluateObligation()*, like this

```
public class PrintObligation extends BaseObligationHandler {

    private static Logger logger = Logger.getLogger(PrintObligation.class);

    public PrintObligation(String id){
        super(id);
    }

    public PrintObligation(String id,int precedence){
        super(id,precedence);
    }

    @Override
    public void evaluateObligation(ObligationProcessingContext arg0,
        ObligationType arg1) throws ObligationProcessingException {
        List<AttributeAssignmentType> attributeAssignmentsList =
            arg1.getAttributeAssignments();
        for(AttributeAssignmentType attributeAssignment : attributeAssignmentsList){
            logger.debug("Id of the obligations is "
                +attributeAssignment.getAttributeId()+"
                and the value is "+attributeAssignment.getValue());
        }
    }
}
```

The **TestObligationService** class is very simple, it defines two instances of the **PrintObligation**, given the obligation id and the number in which we want it to be executed.

```
PrintObligation printGroupId = new PrintObligation("urn:eggee:groupmapping",2);
PrintObligation printUserId = new PrintObligation("urn:eggee:usermapping",1);
```

After this we add the two handlers to the set of handlers in the obligation service.

```
obligationService.addObligationhandler(printGroupId);
obligationService.addObligationhandler(printUserId);
```

Then the XACML Result is created like this and then given as a input parameter for creating a **ObligationProcessingContext**.

```
ResultType result = (ResultType)resultUnmarshaller.
    unmarshall(documentXACMLRequest.getDocumentElement());
ObligationProcessingContext obligationContext = new ObligationProcessingContext(result);
```

XACML Result xml file is located in the input folder and is named XACML-Result.xml, it looks like this

```
<Result xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
    ResourceId="CE.pakgrid.org.pk:2119/jobmanager-lcgpbs-dteam/dteam">
    <Decision>Permit</Decision>
```

```

<Status>
  <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
</Status>
<Obligations xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
  <Obligation FulfillOn="Permit" ObligationId="urn:egge:usermapping">
    <AttributeAssignment AttributeId="UID"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      001
    </AttributeAssignment>
  </Obligation>
  <Obligation FulfillOn="Permit" ObligationId="urn:egge:groupmapping">
    <AttributeAssignment AttributeId="GID"
      DataType="http://www.w3.org/2001/XMLSchema#string">
      005
    </AttributeAssignment>
  </Obligation>
</Obligations>
</Result>

```

The important thing to notice here is to see that the `ObligationId` attribute has the same value as we gave as input to the **PrintObligation** constructor. The last thing we do is call *processObligations* like this

```
obligationService.processObligations(obligationContext);
```

and then the value and the id of the obligations gets printed out.

Running test class sample programs

All output XML examples are generated by the test classes. These test class can interactively generate the most important new elements and un/marshall the new elements, code examples is also from these classes. The program must be run with `-Djava.endorsed.dirs=path/to/endorsed/` option to the VM, inside this directory the files

- xalan.jar
- xercesImpl.jar

should be present, OpenSAML depends on these jar files. Inside the project there is a folder that contains all these files and are named endorsed.

Short summary of the classes:

PrintObligation Extends the `BaseObligationHandler` class for handling obligations. This just prints out the id and value of the obligation

SampleProgram Creates and tests the new classes for the extension, SAML 2.0 profile of XACML, `XACMLPolicyQueryType`, `XACMLAuthzDecisionQueryType`, `XACMLAuthzDecisionStatementType` and `XACMLPolicyStatementType`.

SampleProgramXACMLProvider Provides example on how to use the object providers for the implementation of the XACML specification. This is only implementing the XML elements,NOT PDP and such.

TestObligationService This is a test class for testing the obligation handling code in the package org.opensaml.xacml.provider.

TestSAMLXACMLConverter This shows how to map a `< saml : Attribute >` to a `< xacml : Attribute >`, as written out in the SAML 2.0 Profile of XACML, Working Draft 5.