OSTwoUsrManJavaAnyTypes

Dealing with any Types

A couple of elements within the SAML schema are defined with a special, wildcard, type known as an `anyType` . These elements can take any valid XML as content which translates to the SAMLObjects representing those element allowing any XMLObject as a child. Specifically these are, in SAML 1.X: `Advice`, `SubjectConfirmationData`, `AttributeValue`, and `StatusDetail` and in SAML 2.0: `SubjectConfirmationData`, `Advice`, `AuthnContextDecl`, `AttributeValue`, `Extensions`, `StatusDetail`, `StatusResponse`, and all the metadata endpoint elements.

Adding Content to any Types

Every SAMLObject representing an XML element of type `anyType` extends `org.opensaml.xml.ElementExtensibleXMLObject` which defines the method `getUnknownXMLObjects()` . XMLObjects representing the content to be added to the SAMLObject should be added to the list returned by this method. These child objects will then be marshalled as any other children object.

Building XMLObjects for use as content works a bit differently than the normal method for building objects, because some child elements are expected to have a specific name (e.g. AttributeValue) but can still contain any valid XML. To build the XMLObject in this case get the builder for the XMLObject as you normally would, but instead of invoking `buildObject()` you need to use either `buildObject(String, String, String)` or `builidObject(QName)` to override the default element name. It's also good practice to explicitly state the schema type of this object as it makes it easier for the receiver of the message to parse it. To do this pass in a `QName` that represents the schema type in as an additional, final, argument to the methods just discussed. All the objects in the OpenSAML library, on their interface, define a static `QName` called TYPE_NAME that can be used as this argument.

Reading Content of any Types

The method `getUnknownXMLObjects()` provides access to the unknown content of SAMLObjects defined with the any type. Reading information from this list can be problematic if you're not exactly sure what is in the returned list. If, during the unmarshalling process, the library encountered an element whose xsi:type or element name was not recognized (i.e. was not explicitly listed in the OpenSAML configuration files) the library will unmarshall the element into the `org.opensaml.xml.ElementProxy` XMLObject. No information is lost, but if you were expecting to cast the object to a specific SAMLObject interface you'll receive a class cast exception when you try. If the library understands the element or element type, though, it is safe to cast it to the respecting interface.

AttributeValues

Probably the most commonly encountered example of this is the `AttributeValue` element. There is an AttributeValue interface but nothing implements it, which means you can't cast classes to it. It exists simply to hold name constants which may be used when building other XMLObjects via the `XMLObjectBuilder#buildObject(String, String, String)` or `XMLObjectBuilder#buildObject(QName)` methods.

Here's an example of an `AttributeValue` that carries string content. The `AttributeValue` element will express an `xsi:type` of `xs:string`.

```
// Normally built Attribute object
Attribute attribute;
XSStringBuilder stringBuilder = (XSStringBuilder) Configuration.getBuilderFactory().getBuilder(XSString.
TYPE_NAME);
XSString stringValue = stringBuilder.buildObject(AttributeValue.DEFAULT_ELEMENT_NAME, XSString.TYPE_NAME);
stringValue.setValue("myStringValue");
attribute.getAttributeValues().add(stringValue);
```

Here's an example of an `AttributeValue` that carries element content. An `XSAny` object is used to represent the `AttributeValue`, in order to allow the addition of element children. The `Role` child element in the example is also represented by an `XSAny`. Another and more correct approach would be to implement XML object provider support for the `Role` element.

```
XMLObjectBuilderFactory bf = Configuration.getBuilderFactory();

XMLObjectBuilder<XSAny> xsAnyBuilder = bf.getBuilder(XSAny.TYPE_NAME);

XSAny role = xsAnyBuilder.buildObject("http://www.hhs.gov/healthit/nhin", "Role", "nhin");
role.getUnknownAttributes().put(new QName("code"), "112247003");
role.getUnknownAttributes().put(new QName("codeSystem"), "2.16.840.1.113883.6.96");
role.getUnknownAttributes().put(new QName("codeSystemName"), "SNOMED CT");
role.getUnknownAttributes().put(new QName("displayName"), "Medical doctor");

XSAny roleAttributeValue = xsAnyBuilder.buildObject(AttributeValue.DEFAULT_ELEMENT_NAME);
roleAttributeValue.getUnknownXMLObjects().add(role);

Attribute attribute = (Attribute) bf.getBuilder(Attribute.DEFAULT_ELEMENT_NAME).buildObject(Attribute.
DEFAULT_ELEMENT_NAME);
attribute.setName("UserRole");
attribute.setNameFormat("http://www.hhs.gov/healthit/nhin");
attribute.getAttributeValues().add(roleAttributeValue);
```

When marshalled and serialized, this produces the following XML:

```
<saml2:Attribute xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion" Name="UserRole" NameFormat="http://www.hhs.
gov/healthit/nhin">
  <saml2:AttributeValue>
    <nhin:Role xmlns:nhin="http://www.hhs.gov/healthit/nhin" code="112247003" codeSystem="
2.16.840.1.113883.6.96" codeSystemName="SNOMED CT" displayName="Medical doctor"/>
  </saml2:AttributeValue>
</saml2:Attribute>
```