OSTwoUsrManCPPCreateFromXML

Creating SAML Objects from an XML Source

Creating SAML objects from an XML source is done through a process known as unmarshalling which operates on DOM Elements. The OpenSAML 2 library contains unmarshalling support for known types and elements within its implementation classes, and the appropriate one is selected based off of either the XML Schema type of the Element to be unmarshalled or the Element's QName (a tuple representing the namespace the element is in and the element's name).

Unmarshalling is part of the core XMLObject interface. As a result, you can unmarshall either against a constructed (but empty) XMLObject, or using `XMLObjectBuilder` methods that take DOM elements or DOM documents. The document can be **bound** during this process as well, to control which XMLObject instance is responsible for freeing the document.

Unmarshalling Basics

1. (Optionally) Parse your XML file using `ParserPool::parse()` to construct a DOM representation of the document. Both schema-aware (validating) or non-validating parser pools can be used. The `SAMLConfig` class includes methods for accessing both kinds of pools. The pool API improves efficiency by reusing parser instances for you.
2. Get a builder using the `XMLObjectBuilder::getBuilder(const DOMElement*)` method. The element argument is the root element from which you wish to begin unmarshalling. This is normally the document element if you're unmarshalling data from a file.
3. Invoke the `buildFromElement(DOMElement*)` or `buildFromDocument(DOMDocument*)` method on the returned `XMLObjectBuilder` . Alternatively, just build an empty object and use its unmarshall() methods directly. Usually the builder methods are easier to use.

By default, building from a DOMDocument will **bind** the document to the root object returned, and document ownership is transferred to the object. You must free the resulting XMLObject, but the document will be freed automatically when the bound object is. Building from an element does not **bind** the owning document by default, but this can be overridden.

Unmarshalling Example

The following example shows the parsing and unmarshalling of a SAML 2 metadata document. Exception handling code has been omitted here for readability, but the example does show how to protect the application from leaks by wrapping the DOMDocument in a janitor class before unmarshalling. If that results in an error, the document will be freed for you. Otherwise, the documentis released from the janitor after unmarshalling succeeds.

```
// Parse metadata file
string inCommonMDFile = "/data/org/opensaml/saml2/metadata/InCommon-metadata.xml";
ifstream in(inCommonMDFile);
DOMDocument* inCommonMDDoc = XMLToolingConfig::getConfig()->getParser().parse(in);
XercesJanitor<DOMDocument> janitor(inCommonMDDoc);

// Get apropriate builder.
const XMLObjectBuilder* builder=XMLObjectBuilder::getBuilder(inCommonMDDoc->getDocumentElement());

// Unmarshall using the document, an EntitiesDescriptor in this case.
XMLObject* xo=builder->buildFromDocument(inCommonMDDoc);
janitor.release();
EntitiesDescriptor* inCommonMD = dynamic_cast<EntitiesDescriptor*>(xo);
```

After all that, only `inCommonMD` (or `xo`) needs to be deleted when you're done with it. If an exception is thrown, nothing will leak.

Additional Unmarshalling Information

- When fetching a builder based on an element, the system first checks to see if the element has a schema type specified by an `xsi:type` attribute. If it does, the factory attempts to lookup a builder for that schema type. If the element does not have a specified schema type, or no builder is registered for that schema type, the system uses the element's name.
- SAML objects include a custom type-safe builder API that includes a static method that can return a new empty instance of a given SAML object. This can be used in place of the multi-step builder lookup, if you prefer to just unmarshall against the resulting empty object.
- All SAML Objects you unmarshall contain a reference to their DOM. This cached DOM will be destroyed if the SAML Object is changed in some way (e.g. a child element is removed) and developers should never attempt to directly manipulate it. It does, however, make the marshalling of objects significantly faster and allows things like the digital signatures and encrypted information to be preserved, both of which make dealing with SAML inside of SOAP much nicer.