OSTwoUsrManCPPCreateFromScratch

Creating SAML Objects from Scratch

New SAML Objects are created through a builder class. This class is responsible for creating an object using some particular implementation and setting up some initial information such as element name and schema type.

Building Basics

1. Using `XMLObjectBuilder` static methods, you can lookup builders by element or QName, but each known type/element `Foo` also supports a type-safe `FooBuilder` interface that allows a single new object of that type to be created. You should use the static `FooBuilder::buildFoo()` method unless you're building multiple instances. Then it's more efficient to ask for a const pointer to a builder and reuse it.
2. If you're using the builder directly, invoke the appropriate `buildObject()` method on the returned `FooBuilder` . Normally the no-argument method is sufficient but other methods allow you to explicitly set the element name and schema type of the constructed object.

Building Example

The following example demonstrates creating a SAML 1.1 Assertion from scratch. The exception handling code has been omitted here for readability.

```
// Get the assertion builder based on the assertion element QName
const AssertionBuilder* builder = dynamic_cast<const AssertionBuilder*>(
       XMLObjectBuilder::getBuilder(QName(SAMLConstants::SAML_NS,Assertion::LOCAL_NAME))
);

// Create the assertion
Assertion* assertion = builder->buildObject();
```

A simpler way to build one object is below. This simply runs the same code internally along with the necessary casts. Builder lookup and casting is expensive, so you should do it yourself if you plan to reuse the builder.

```
// Create the assertion
Assertion* assertion = AssertionBuilder::buildAssertion();
```

This more advanced example uses the schema type name and builds an element of the Assertion schema type but with a different name in a different name space. You might do this to reuse an assertion's content model in some other element that permits it. Note that this doesn't **nest** an assertion inside that element, it literally gives you an assertion with a different root element name.

```
// Get the assertion builder based on the assertion element QName
const AssertionBuilder* builder = dynamic_cast<const AssertionBuilder*>(
       XMLObjectBuilder::getBuilder(QName(SAMLConstants::SAML_NS,Assertion::LOCAL_NAME))
);

// Create the assertion
QName type(SAMLConstants::SAML_NS,Assertion::TYPE_NAME);
Assertion* assertion = builder->buildObject(
              someOtherNamespaceURI, MyAssertionElementName, someOtherNamespacePrefix, &type
);
```

Note that string literals cannot be used in C++ because only the UTF-16 encoding is supported by Xerces natively. More about this in the OSTwoUsrManCPPStrings topic.

Additional Building Information

- While every type-specific FooBuilder has a no-argument buildObject method and a static `buildFoo` method, the more generic XMLObjectBuilder interface does not have this method. That's why the additional casting is needed.
- Builders are thread-safe and stateless. You may use an instance to create any number of objects.

%COMMENT%