ExtensionInterfacesFeature

Extension Interfaces Feature

Warning Advanced feature.

This feature enables a user to specify sets of generated xbean interfaces that will extend user interfaces (extension interfaces). Also, generated implementation classes will also implement the methods in the extension interfaces, they will delegate to a user handler.

This feature is useful for extending the xbeans with methods unrelated to the properties in the schema type it represents. For example to a `PurchaseOrder` type a `float getTotalAmount()` method can be added, which will calculate the total amount of the ordered items.

Or this feature will enable the xbean objects to represent other interfaces.

Example

Given a schema that generates for example `xsd.company.CompanyDocument` xbean, a user might want it to extend his own interface `myPackage.Foo`.

Configuration

The .xsdconfig file that will enable this is:

```
<xb:config xmlns:xb="http://xml.apache.org/xmlbeans/2004/02/xbean/config">
...
    <xb:extension for="xsd.company.CompanyDocument">
        <xb:interface name="myPackage.Foo">
            <xb:staticHandler>myPackage.FooHandler</xb:staticHandler>
        </xb:interface>
    </xb:extension>
</xb:config>
```

The *for* attribute can accept a list of xbean java interfaces (separated by space) or ***** to include all of them in the extension. Why the java names of the xbeans, when all we have at the beginning is schema files? Because there are java xbeans generated also for anonymous types, and we believe that the java names is a more cleaner solution than inventing an expression language for specifying all the anonymous/implied schema types.

Extension Interface

The generated xbean interface `xsd.company.CompanyDocument` will extend `myPachage.Foo` interface. In our example the interface `myPackage.Foo` contains only one method:

```
    String foo(String s);
```

Extension Handler

In the xbean implementation class `xsd.company.impl.CompanyDocumentImpl` methods will get generated to implement the extension interface and they will delegate to the extension handler methods.

The handler `myPackage.FooHandler` has to contain a public static method with the same name as the interface method, with the first parameter of type XmlObject, followed by the parameters of the interface method:

```
    public static String foo(XmlObject xo, String s)
    {
        return "{in FooHandler.handleFoo(s: " + s + ", xo: " + xo + ")}";
    }
```

This method will be executed every time the foo method on `xsd.company.CompanyDocument` will be called.

Building

Because of the circular dependency building all the pieces can be a little tricky. (An implementation using JAM will improve the building process.)

1. scomp .xsd files to xmltypes.jar (.xsdconfig files might be included but they should not contain an `extension` element).

```
scomp src\company.xsd
```

2. write and compile the extension interfaces and the handler classes using xbean.jar and xmltypes.jar

```
javac -d build\classes -classpath build\ar\xbean.jar;xmltypes.jar src\myPackage\*.java
```

3. add the `extension` element to the .xsdconfig file
4. run scomp again with .xsd and .xsdconfig files setting xmltypes.jar and compiled extension classes on the classpath

```
java -classpath build\ar\xbean.jar;xmltypes.jar;build\classes org.apache.xmlbeans.impl.tool.
SchemaCompiler src
```