CoreXml

Table of contents

1. Overview

CoreXml currently has the following subcomponents.

- **SimpleConfig** is a facility for interfacing an XML configuration file to a program
- **context** allows program components to share variable definitions through the use of (possibly nested) symbol tables
- **expr** provides limited XPath 1.0 expression support
- **bind** is a data binding facility allowing programmers to describe an XML file with a data structure which automates the production of Java objects from the XML and vice versa.
- **om** is CoreXml's Object Model for XML

Each of these is documented in the Javadocs, which can be reviewed by clicking on **API** in the menu to the left. **Source** code is also available on this Web site.

2. SimpleConfig

This is used to import values from an XML configuration file into a Java program.

The configuration file

- may have an XML declaration
- may contain comments
- must have one top-level element
- and may contain any number of simple subelements

`SimpleConfig.bind(obj, reader)` reads the configuration file; discards any XML declaration, comments, and the top-level element; and then uses each subelement to set a value in the object. It does this by assuming that each subelement tag has a corresponding setter in the object's class. So if, for example, the subelements include

```
<amp>2.0</amp>
<volt>115</volt>
<hertz>60</hertz>
```

then `bind()` will in effect make a series of mactching calls

```
obj.setAmp(2);
obj.setVolt(115);
obj.setHertz(60);
```

to set field values in the object.

Attributes are ignored. Values may only be set in one object, although any number of distinct values may be set. If you need something more powerful, use `corexml.bind` , below.

3. Context

A context is a symbol table. Given a context, you can then add symbols and definitions to it.

```
Context context = new Context();
    context.bind("abc", "def")
           .bind("ghi", "jkl")
           .bind("mno", "pqr");
```

The context can then be passed to other program modules and the symbols retrieved by a simple lookup call.

```
String value = context.lookup(name);
```

You can build context hierarchies:

```
Context a = new Context().bind("abc", "dog's dinner");
Context b = new Context(a).bind ("def", "jolly good fellow");
```

In this case, because context `a` is the parent of `b` , a `b.lookup("abc")` will return "dog's dinner", because when a lookup in context `b` fails the search automatically continues in the parent.

What gives contexts real power is the possibility of using them to perform symbol substition in texts. If you are familiar with Ant, then you have used this. CoreXml's `context` package allows you to automatically convert text containing variables, symbolic expressions like `${abc}` . Anything found between the opening `${` and the closing `}` is regarded as a variable, is looked up in the context, and replaced with its value.

```
Context a = new Context( ).bind("abc", "dog's dinner")
                          .bind("def jolly good fellow", "dunno who");
Context b = new Context(a).bind ("def", "jolly good fellow");
String s = new Expr("${abc} is a ${def}").resolve(b);
```

The `Expr(s)` constructor parses `s` into a series of literals and Symbols. `resolve(context)` then uses the `context` to replace the symbols with their values, recursing as necessary.

As you might expect, `s` has become "dog's dinner is a jolly good fellow".

Symbols can be nested, making it possible to write things like

```
<abc attr="${abc ${def}}">
```

When this is evaluated, first the inner symbol, `${def}` , will be evaluated, producing, in this case, `"jolly good fellow"` . This will then be concatenated with the literal preceding it to produce `"${abc jolly good fellow}"` . The software will then look this up in the context, replacing the entire substring with its match, resulting in

```
<abc attr="dunno who">
```

Contexts are particularly useful in conjunction with `corexml.expr` XPath expressions.

4. Expr

`corexml.expr` is a partial implementation of the W3C's **XPath** expression language. This interprets expressions

- relative to the current Node
- and against a specified Context

Expressions are singly or doubly quoted strings. They may contain *steps;* a common example would be the familiar "../.." meaning 'something two levels above the current node'. They can also contain variables to be resolved against the context, such as "${partNumber}".

XPath has four data types:

- **boolean**
- **number**
- **string**
- **node set**

There are rules for casting these types to one another, but none of the other types can be cast to a NodeSet. Some of the rules for casting are less than intuitive. The empty string, for example, casts to `false,` but any other string to `true,` so both boolean("true") and boolean("false") resolve to `true.` The numeric type is equivalent to Java's Double. That is, it is an object.

`corexml.expr` will correctly parse all XPath expressions, but only a fraction of the standard library functions have been implemented and only steps along the child axis will be evaluated correctly.

XPath is expected to be used by the XLattice project primarily in

- extracting data from specification documents and
- generating Web pages and other such documents

We will be coding more of XPath as necessary to meet these ends.

5. Bind

This is a Java data binding facility.

XML documents are bound to instances of Java classes by writing what amounts to a description of the binding in terms of a number of classes:

- a **Mapping,** which relates the top-level XML element to the topmost Java class instance
- **SubMappings,** used where a subelement maps into a child object
- **Collectors,** signalling the presence of a element in the XML which does not cause the creation of a Java object on input, and
- **Bindings,** which relate XML values to values of fields in objects

There are several types of bindings.

- **AttrBindings,** connecting attributes to fields
- **EmptyElBindings,** which set a boolean to true if a subelement is present
- **SubElBindings,** which connect the value of a subelement to a field value, and
- **TextBindings,** relating the text within a subelement to a String field in the corresponding object

The constituents may be either optional or may be allowed to repeat.

Additional descriptor classes will be added after more experience with the package in its current form. We also intend to add a facility for generating the mapping from an XML description.

6. XLattice XML Object Model

All of the above functionality, except for `corexml.context,` is implemented using `corexml.om,` XLattice's XML object model and the XML pull parser.

The primary objects are

- **Node,** which the objects that follow extend
- **Document**
- **DocumentType,** which is stubbed
- **Element**
- **Holder,** a superclass for *Document* and *Element*
- **Attr**
- **Comment**
- **Text**
- **Cdata,** which extends *Text* , and
- **ProcessingInstruction**

There are also a number of containers:

- **NodeList,** used by *Holders* to contain child Nodes
- **AttrList,** used by *Elements* to contain attributes

While this intention has not been fully realized, the objective has been to provide a set of constructors and related methods which are uniform in their use and will only generate internally consistent objects. As shortcomings in this direction are identified, they will be corrected.

The **XmlParser** class transforms an XML document into a tree of *Nodes* with a *Document* at its root. Each of the *Node* subclasses has a `toXml()` method which recursively generates a String representation of the *Node* and its descendents. This means that any Document `doc` can be converted to XML text by invoking `doc.toXml()`.

For convenience in writing unit tests, JUnit's `TestCase` class has been extended. `org.xlattice.corexml.CoreXmlTestCase` has a `assertSameSerialization(String s1, String s2)` method which ignores whitespace while comparing `s1` and `s2`.

The Javadocs should be helpful for those needing to use or modify any of this code. Thorough unit tests exist for all of these classes, providing very detailed examples of how to use the software.