UTF-X Manual
Unit Testing Framework - XSLT

Jacek Radajewski
Alexander Daniel
Release 0.0.8
26 August 2006

1 Introduction

1.1 What is 'unit testing'?

Unit testing is a method of testing software modules such as functions and methods in isolation from other modules. One or more unit tests is normally written for each software module. Each unit test only tests one unit of functionality. A collection of unit tests for a software project forms a regression test. Regression test when executed runs all unit tests and tests the software system for regression in the software. If all unit tests pass then the regression test also passes.

1.2 What is UTF-X?

UTF-X is an extension to the JUnit Java unit testing framework and provides functionality for unit testing XSLT stylesheets. UTF-X strongly supports the test-first-design principle with test rendition and test validation features allowing you to visually design your test before you start working on the on stylesheet. UTF-X was originally built to test XSLT stylesheet used in an XML publishing system so it has good support for DTD validation, XHTML and XSL:FO stylesheets.

2 Quick Start

Currently this is the only 'getting up and running' documentation available. Future releases will ship with more detailed documentation.

2.1 Installing UTF-X

1. Ensure you have JDK 5.0 installed on your computer. UTF-X uses Java's new language features and will not run or compile on older versions of Java. You can download Java 5.0 software development kit either from Sun Microsystems (preferred) or from BEA Systems.

2. If you are planning on compiling UTF-X or using the ant build file (recommended) you will also need to have ant 1.6.x installed. You can obtain

ant from http://ant.apache.org.

3. Download the latest version of UTF-X from SourceForge.

4. Unzip the file in a suitable location.

5. Run the samples by typing `ant samples`.

2.2 Creating Tests

2.2.1 Generating tests

UTF-X ships with a test generator which can be used to generate Test Definition Files (TDFs) from existing XSLT stylesheets.

`utfx.testgen.TestGenerator` accepts a `-xslt xslt_filename` argument which specifies the path to the stylesheet from which we are generating the test. The test file will be generated in a directory called 'test' under the directory where the stylesheet resides. If this directory does not exist then it will be created. The TestGenerator will not overwrite existing test files unless you pass `-f` as an argument to the program. For example, to generate a TDF for a stylesheet `c:\xsl\webpage.xsl` we would execute:

```
java -cp build/jar/utfx.jar utfx.testgen.TestGenerator -xslt c:\xsl\webpage.xsl
```

which would create a directory `c:\xsl\test` and a TDF `c:\xsl\test\webpage_test.xml`

2.2.2 Creating tests from scratch

We suggest you look at the samples provided with the distribution. You'll find these in the samples directory. This manual has been created using one of those samples (utfxdoc).

See section 4 Test Definition File Structure.

2.3 Running Tests

UTF-X currently does not provide any nice wrapper scripts or GUI front end applications which you can use to run the software. You have the following options:

1. run `utfx.runner.TestRunner` from the command line ensuring that `utfx.jar` is in your classpath. You will also need to pass in either `utfx.test.dir` or `utfx.test.file` property to the JVM and tell the TestRunner that the test class is `utfx.framework.XSLTRegressionTest`. So in order to run UTF-X tests under `c:\myxslt_tests` you would execute `java -cp`

```
build/jar/utfx.jar -Dutfx.test.dir=c:\myxslt_tests
utfx.runner.TestRunner utfx.framework.XSLTRegressionTest
```

2. Run UTF-X tests from an ant build file. See target 'samples' in `build.xml` in the project tree.

3. Run UTF-X tests within your existing JUnit configuration. Use `utfx.framework.XSLTRegressionTest` class as the test suite. You will need to pass in either `utfx.test.dir` or `utfx.test.file` property to the JVM.

Remember that you WILL need Java 5.0!

3 Features

UTF-X 0.0.8 supports the following features:

1. XSLT transformation test case. This is achieved by passing the contents of the `<utfx:source>` fragment through the XSLT stylesheet that is being tested and asserting that the result is the same as the contents of the `<utfx:expected>` fragment. Other types of assertions may be added in future releases.

2. DTD that specifies the structure of Test Definition Files (TDFs). This feature allows for TDF validation and easy TDF authoring when using a DTD aware XML editor.

3. `<utfx:source>` and `<utfx:expected>` fragment validation using a DTD (no schema support yet). If the `validate` attribute is set to `yes` on either of the two fragments then the framework will automatically create a corresponding validation test case and add it to the test file suite.

4. TDF rendition (XHTML and XSL:FO). You can render test definition files to produce XHTML or XSL:FO output and therefore visually inspect your tests. This feature is particularly useful if you like the idea of test-first-design. It allows you to write all the tests before you start on the stylesheet.

5. Automatic test file generation from existing XSLT stylesheets with `utfx.testgen.TestGenerator`

6. Pluggable test file filters for locating test definition files. See the `utfx.testfile-filter.class` property in the `utfx.properties` file for examples.

7. Pluggable parsers/pre-processors for `<utfx:source>` fragment parsing (`utfx.framework.SourceParser`). See the `utfx.source-parser.class` property in the `utfx.properties` file for examples.

8. Pluggable result printers for formatting test results. UTF-X ships with a number of result printers including one for XML and one for ANSI colour terminals (e.g. Linux xterm). See the `utfx.result-printer.class` and

`utfx.result-printer.output-file` properties in the `utfx.properties` file for examples.

9. Absolute XPath expressions (e.g. `select="/tree"`) work as expected in the stylesheet under test. Additionally one can set the context node for a test with the `context-node` attribute in the `utfx:source` element. Note: context-node="/" can be used for named templates but not for match templates because it will recurse infinitly.

10. Pluggable XSLT engines. This is achieved by setting standard JAXP `javax.xml.transform.TransformerFactory` property. UTF-X 0.0.8 has been tested with Saxon 8.7 and Apache Xalan 2.6.0 Interpretive processor. JDK 5.0 internal Xalan XSLTC and Apache Xalan 2.6.0 XSLTC have been tested as well and everything works fine but absolute XPath expressions because of Xalan issue 1928.

11. UTF-X 0.0.8 has been tested on Windows XP, Linux and Mac OS X platforms.

4 Test Definition File Structure

This section looks at the structure of a UTF-X test definition file.

4.1 DOCTYPE

DOCTYPE declarations are not required by the UTF-X framework and are only useful if a DTD aware editor is used to author the TDFs. DOCTYPE declaration can be either a simple reference to the UTF-X test DTD like this:

```
<!DOCTYPE utfx:tests PUBLIC "-//UTF-X//DTD utfx-tests 1.0//EN" "utfx_tests.dtd">
```

or a more complex one that also includes references to DTDs for the XML source and generated target:

```
<!DOCTYPE utfx:tests PUBLIC "-//UTF-X//DTD utfx-tests 1.0//EN" "utfx_tests.dtd" [

<!ENTITY % utfxdoc PUBLIC "-//UTF-X//DTD utfxdoc 1.0//EN" "utfxdoc.dtd">
        %utfxdoc;

<!ENTITY % xhtml1_dtd PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"xhtml1-transitional.dtd">
    <!-- this must be commented out during test run; see issue 12 -->
    <!-- %xhtml1_dtd; -->
]>
```

In the second case your XML editor will become aware of not only the structure of TDFs, but also the structure of your `<utfx:source>` and `<utfx:expected>` fragments. Authoring tests in this mode is much easier as the editor will ensure that both source and expected fragments are valid. Not all XML editors support internal declarations.

4.2 Stylesheet

Immediately following the root element is a `<utfx:stylesheet>` element which provides a pointer to the XSLT stylesheet that is being tested. UTF-X currently assumes that the stylesheet is in a directory above the one where the test resides. For example, if your TDF is located in directory `c:\myxsl\test\` and you specify that the stylesheet under test is `my_stylesheet.xsl` then UTF-X will assume you are refering to `c:\myxsl\my_stylesheet.xsl` stylesheet.

```
<utfx:stylesheet src="my_stylesheet.xsl" />
```

Figure 4.1: Specifying the stylesheet under test.

4.3 Source and expected validation

If you want to enable source and expected fragment validation you must specify system and public identifiers. These identifiers are used to form the DOCTYPE declaration during the validation of each source and expected fragment.

```
<utfx:source-validation>
  <utfx:dtd public="-//UTF-X//DTD utfxdoc 1.0//EN" system="utfxdoc.dtd" />
</utfx:source-validation><utfx:expected-validation>
  <utfx:dtd public="-//W3C//DTD XHTML 1.0 Transitional//EN"
system="xhtml1-transitional.dtd" />
</utfx:expected-validation>
```

4.4 UTF-X test case

Let us now look at the structure of an actual test case. Each test case must have a name followed by `<utfx:assert-equals>`. Assert-equals has two required elements which are the `<utfx:source>` and `<utfx:expected>` fragments.

```
<utfx:test>
  <utfx:name>sect1 with title only</utfx:name>
  <utfx:assert-equal>
    <utfx:source validate="yes">
      <section id="section1">
        <heading>Section 1</heading>
      </section>
    </utfx:source>
    <utfx:expected validate="yes">
      <a name="section1" />
      <h1>Section 1</h1>
    </utfx:expected>
  </utfx:assert-equal>
</utfx:test>
```

Figure 4.2: UTF-X Test

UTF-X executes a test by passing the contents of `<utfx:source>` through the XSLT stylesheet and comparing the result with the contents of `<utfx:expected>`. If both are equivalent XML fragments then the test passes. Otherwise the test fails.

4.5 Passing parameters to named templates

Parameters can be passed to named templates like in XSLT with the `utfx:with-param` element.

```
<utfx:test>
    <utfx:name>named template with parameter</utfx:name>
    <utfx:call-template name="named-template-with-param">
        <utfx:with-param name="a" select="1" />
    </utfx:call-template>
    <utfx:assert-equal>
        <utfx:source>
            <empty />
        </utfx:source>
        <utfx:expected>
            <a>1</a>
        </utfx:expected>
    </utfx:assert-equal>
</utfx:test>
```

Figure 4.3: UTF-X Test of named template with parameters

4.6 Stylesheet parameters

With the `utfx:stylesheet-params` element parameters can be passed to the stylesheet under test. It can also be used to define parameters in a stylesheet. This feature can be very useful for stylesheets which are normally imported by other stylesheets and which expect certain parameters or variables. These stylesheets won't compile unless we define the parameters in the TDF.

```
<utfx:test>
    <utfx:name>stylesheet parameter test</utfx:name>
    <utfx:stylesheet-params>
        <utfx:with-param name="stylesheet-param1" select="'UTF-X'" />
    </utfx:stylesheet-params>
    <utfx:assert-equal>
        <utfx:source>
            <print-param1 />
        </utfx:source>
        <utfx:expected>
            <stylesheet-param1>UTF-X</stylesheet-param1>
        </utfx:expected>
    </utfx:assert-equal>
</utfx:test>
```

Figure 4.4: UTF-X Test with stylesheet parameters

5 Limitations and Known Issues

- UTF-X 0.0.8 has been tested with UTF-8 encoded XSLT stylesheets and Test Definition Files. UTF-X 0.0.8 may work incorrectly if other character encoding formats are used. A unit test which tests the DOMWriter with UTF-16 encoding currently produces an error on windows platform.

- `<xsl:template match="/">` can't be tested because UTF-X uses this matcher for the internal wrapper document when executing the tests.

- Absolute XPath expressions in the stylesheet under test don't work with JDK 5.0 internal Xalan XSLTC and Apache Xalan 2.6.0 XSLTC because of Xalan issue 1928.

- For a full list of unresolved issues please check UTF-X issue tracker.

6 Resources

- JUnit is the core unit testing framework used by UTF-X. JUnit provides unit testing functionality for the Java programming language.

- XSLTunit is one of the first XSLT unit testing tools. Software development group at the USQ began the UTF-X project because they found XSLTunit too cumbersome to use.

- Juxy is another XSLT unit testing framework written in Java.

- <oXygen/> is an XML editor that supports internal declarations such as those used in UTF-X samples. All UTF-X XML documents and XSLT stylesheets have been authored using <oXygen> xml editor.

Glossary

- **Unit Test** - A test which tests a single piece of functionality in a single software module. There is usually one or more unit tests for each software module (function, method or template)
  http://www.extremeprogramming.org/rules/unittests.html

- **Regression Test** - An automated test consisting of many unit test. Regression tests are executed after changes to a software system have been bade to ensure that no regression (introduction of bugs or loss of functionality) has been made.

- **UTF-X** - Unit Testing Framework - XSLT

- **XML** - eXtensible Markup Language

- **XSL** - eXtensible Stylesheet Language

- **XSLT** - XSL Transformations. XSLT is a functional language designed to transform XML documents into other documents (most likely also XML). The most common use of XSLT is in transformation of XML documents into HTML or XHTML.

- **XSL:FO** - XSL: Formatting Objects is an XML based typesetting language specially designed to be used with XML and XSLT. XSL:FO is usually generated from an XML document by a means of a XSLT stylesheet and then processed through a XSL:FO processor to obtain a rendition, usually in the form of a PDF or PostScript document.

- **JAXP** - Java API for XML Processing

- **DOM** - Document Object Model

- **SAX** - Simple API for XML processing

- **XSL:FO processor** - or XSL:FO engine, is a software for transforming XSL:FO documents into human readable rendition, usually page description languages such as PDF or PostScript.

- **Document Fragment** - lightweight DOM Document. One of the most important differences between a Document and Document Fragment is that unlike a Document a Document Fragment does NOT have to be well formed in the sense that it does not need to have ONE root node. Good examples of Document Fragments are DOM representations of the contents of `<utfx:source>` and `<utfx:expected>` elements.

- **TDF** - Test Definition File. An XML file that defines a suite of UTF-X tests for a specified XSLT stylesheet.