

Introduction to DeltaXML

Introduction to DeltaXML

Copyright © 2005 DeltaXML Ltd.

Table of Contents

1. Getting Started	1
1.1. Overview	1
1.2. Installing DeltaXML	1
1.3. Checking your installation	2
1.3.1. A Note About Windows and Unix Paths	3
1.4. Tutorial navigation	3
1.5. Acknowledgements	3
2. Generating a Delta - Structured Diff	4
2.1. Changes Only	4
2.2. Full Delta	4
2.3. Code sample - Simple.java	4
3. Applying a Delta - Structured Patch	6
3.1. Forward combination	6
3.2. Reverse combination	6
3.3. Code sample - Applying delta patches	6
4. Viewing the changes	8
4.1. Generating XHTML with XSL	8
4.2. DeltaWing - Building a Java UI	9
4.3. Advanced visual display options	10
5. Working with DeltaXML	12
5.1. The DeltaXML delta format	12
5.2. Managing white space	12
5.3. Namespaces	12
6. Distributing your application	14
6.1. Requirements for distribution	14
7. Troubleshooting	15
7.1. SAX errors - invalid XML	15
7.2. XSL errors	15
7.3. DeltaXML error codes	15
8. Further Reading	16
8.1. DeltaXML Documentation	16
8.2. External Resources	16
8.3. Useful Books	16

Chapter 1. Getting Started

1.1. Overview

DeltaXML is a sophisticated XML change control technology. This short tutorial will enable you to use this powerful toolset to solve a wide range of problems.

By working quickly through this tutorial, you'll become familiar with XML comparison terminology, and with features such as "full delta comparison", "orderless comparison" and "input/output filters". You may also want to work through the code examples and experiment for yourself as you read the tutorial.

Further information about DeltaXML is available at the DeltaXML site. If you'd like to see how DeltaXML can offer solutions you may not have considered, such as increased security and database synchronization, you may want to check our Use Cases pages, which present such challenges from various perspectives and discuss how DeltaXML can bring immediate benefits.

Navigating through the tutorial is easy:

- Select Next and Previous to move forward and backward through the tutorial.
- When you're finished with a section, select Next section for the next section. Within a section, use the Section menu button to see the contents of that section. You can return to the main menu at any time by clicking the Main menu button.
- If you'd like to tell us what you think, or if you have a question for the author about the content of the tutorial, use the Feedback button.

When you've completed this tutorial, you'll have DeltaXML installed on your machine, and be ready to undertake configuration for any customizations you require. We hope you find using the DeltaXML Core API both productive and enjoyable.

1.2. Installing DeltaXML

In order to use the DeltaXML Core API you need to install a Java 2 SDK, 1.2.1 or later. We recommend that you use version 1.4.2 or above to ensure maximum efficiency during XML processing. For deployment you will just need a Java2 JVM (JRE). You can download versions from:

- Sun - most popular choice, highly recommended
- IBM developerWorks - may give better performance on some systems
- BEA JRockit - claims excellent performance, used for WebLogic
- Excelsior JET - for Microsoft Windows(TM) platforms
- Sun's Java ports page - for other platforms.

The DeltaXML Core API is distributed in a single ZIP file, containing:

- `deltaxml.jar` - the DeltaXML Core API

- `command.jar` - a command-line controlled version of the DeltaXML engine
- Xerces SAX Parser and Saxon XSLT processor JAR files (`xercesImpl.jar`, `xml-apis.jar` and `saxon.jar`). If you use versions 1.2 or 1.3 of the JDK you will need to use these archives (under the terms of the appropriate Licenses), or replace them with another JAXP 1.1-compliant parser and transformer, to provide parsing and transformation support
- a `samples` subdirectory containing example applications and test files. These are accompanied with various `build.xml` Ant scripts which can be used for compilation and running.
- a `source.zip` package containing the source code to the `command.jar` and `deltawing.jar` commandline and GUI test/driver applications and for various Java based filters.
- a subdirectory containing Javadoc programmers' documentation for the Core API (this documentation can also be found online at the DeltaXML.com site)

Unzip the archive to your hard disk - we will refer to the directory it creates (such as "DeltaXMLCore-3_0_0") as the "DeltaXML Core installation directory", or just "the install dir". Since you'll be accessing DeltaXML Core from here, we suggest a standard location such as `C:\javalibs` for Windows and `/usr/local` or `/opt` for Unix.

As we provide Ant build files to compile and run all demonstration programs you will complete this tutorial faster if you have installed Ant on your machine. Ant can be downloaded from <http://ant.apache.org/>.

1.3. Checking your installation

To make sure you've installed the DeltaXML Core API correctly, follow these steps:

- Go to a command prompt.
- At the prompt, change to the `samples` subdirectory of the install dir.
- If you have Apache Ant installed, just type the command `ant`. This will compile and run the three classes provided as sample code, generating five difference reports for you to examine. Look for the generated XML and HTML files, which should match those provided on our website.
- If Ant has not been installed on your system, some typing is required! First compile `SimpleFiles.java`. For users of JDK 1.4 and above the command will take the form:

```
javac -classpath ..\..\deltaxml.jar SimpleFiles.java
```

If you are running an older version of Java, such as JDK 1.3 or JDK 1.2, you will need to add calls to the parsing and transformation tools to the `-classpath` statement identifying where the DeltaXML Java archive file has been installed to this and subsequent commands. Examples of the form these `-classpath` statements should take are provided in the `README.txt` file in the `samples` directory.

To test the compiled program using JDK 1.4 enter the following command:

```
java -classpath ;..\..\deltaxml.jar;..\..\saxon.jar;..\..\xercesImpl.jar SimpleFiles
```

If everything has been installed correctly, this will create a DeltaXML delta file, `out.xml`.

With the DeltaXMLCore API installed and tested, you're now ready to test its remarkable capabilities.

1.3.1. A Note About Windows and Unix Paths

From the above you'll see that Windows uses "\" and ";", whereas the various Unix flavours use "/" and ":". (In fact you can use "/" with CLASSPATH on Windows too.) To assist brevity, we will from now on use Unix-style only. Note that Ant removes this inconsistency and uses the appropriate conventions for the current platform.

1.4. Tutorial navigation

Navigating through the tutorial is easy:

- Select Next and Previous to move forward and backward through the tutorial.
- When you're finished with a section, select Next section for the next section. Within a section, use the Section menu button to see the contents of that section. You can return to the main menu at any time by clicking the Main menu button.
- If you'd like to tell us what you think, or if you have a question for the author about the content of the tutorial, use the Feedback button.

1.5. Acknowledgements

The success of XML is driven by the desire for open standards for simple and powerful technologies. While our core algorithms and implementation represent a considerable investment in R&D, we are only able to bring them to you in this form thanks to the work of many others. In particular we'd like to say "thank you" to:

- The Apache XML Project [<http://xml.apache.org>] for consistently delivering industrial grade implementations
- The many Java-based projects that have enabled such a rapid improvement in engineering practises and driven XML adoption, such as dom4j [<http://www.dom4j.org>], Apache Ant [<http://ant.apache.org/>] and JUnit [<http://www.junit.org/>]
- DocBook [<http://www.oasis-open.org/docbook/>] and all those involved with DocBook which is used to create this tutorial set.

This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>) - though you can use any JAXP [<http://java.sun.com/xml/jaxp/>]-compliant technology.

Chapter 2. Generating a Delta - Structured Diff

2.1. Changes Only

The `out.xml` file created using the `SimpleFiles` class set up to test the installation of the DeltaXML Core API is an example of a "changes-only" delta file.

A "changes only" comparison retains the structure of the original documents in the delta file, but only includes data that has changed. Markup in the `deltaxml:namespace` indicates additions, deletions and changes so that processing of the delta is very straightforward.

This is the simplest form of comparison and is the default. This delivers the fastest and smallest deltas - if you have access to either of the original documents and the "changes only" delta, you have everything you need for further processing.

See The DeltaXML delta format [</tutorial/dxtutor/dxtutor-6-4.html>] for details of the delta syntax used by DeltaXML.

2.2. Full Delta

A Full Delta comparison generates a delta with the same structure as the Changes Only format, but also including all data from the original documents. Since *all* data from *both* documents is included, marked up to show changes, this is the preferred format for visual display.

To select a Full Delta, set the `isFullDelta` feature to `true` - see Setting API features and properties.

An example of an HTML visualization of Full Delta comparison is generated by our `DeltaWing` - Building a Java UI demonstration.

2.3. Code sample - Simple.java

The source code for the `Simple` class, `SimpleFiles.java`, can be found in the `samples` directory. This code shows how a DeltaXML `PipelinedComparator` can be used to compare two XML files specified on the command line, returning the delta to a result file identified by the third argument

```
import com.deltaxml.core.PipelinedComparator;
import com.deltaxml.core.PipelinedComparatorException;
import java.io.File;
import java.io.FileNotFoundException;
import javax.xml.transform.OutputKeys;

/**
 * Demonstrates the construction of a very simple
 * pipeline doing fileIO.
 */
public class SimpleFiles {

    public static void main(String[] args)
        throws FileNotFoundException, PipelinedComparatorException
    {
        PipelinedComparator pc= new PipelinedComparator();
        pc.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION, "yes");
    }
}
```

```
    pc.compare(new File("a.xml"), new File("b.xml"), new File("out.xml"));
  }
}
```

Executing this class on two simple XML files will generate a delta file of the changes between the two files. For example, given the files

a.xml:

```
<root>
  <a/>
  <b/>
  <c/>
</root>
```

and b.xml:

```
<root>
  <a/>
  <x/><b/>
  <c attr="hello world"/>
</root>
```

Executed using the command:

```
java -classpath ../..\..\deltaxml.jar;..\..\saxon.jar;..\..\xercesImpl.jar SimpleFi
```

We can generate the following *delta*:

```
<root xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
  deltaxml:delta="WFmodify">
  <a deltaxml:delta="unchanged"/>
  <x deltaxml:delta="add"/>
  <b deltaxml:delta="unchanged"/>
  <c deltaxml:delta="WFmodify" deltaxml:new-attributes="attr=&#34;hello world&#34;
</root>
```

This delta file indicates that the element <x> was *added* and that there was a new attribute for the element <c>.

Chapter 3. Applying a Delta - Structured Patch

3.1. Forward combination

Delta files can be combined with either of the originating files to recreate the other.

The forward combination option of the DeltaXML Core API is the XML equivalent of the Unix/GNU patch utility, which allows you to store just the first original file and the delta - the second, edited, file can be regenerated when required from this pair. This is the default operation of the DeltaXML Combiner.

3.2. Reverse combination

Reverse combination is symmetrical with Section 3.1, "Forward combination" and allows the delta to be combined with the second, edited, file to recreate the original file.

A reverse combination is selected by setting the `isCombineForward` feature to `false` - see Setting API features and properties.

3.3. Code sample - Applying delta patches

In this example an `XMLCombiner` is used to apply a delta to one original file to generate the other. As in the Simple example, this sample works on files specified as command-line arguments. The parameters passed to the `combine` method are:

- `base` - one of the original documents
- `delta` - the delta ("patch") file to be applied
- `output` - the original document to be recreated by DeltaXML.

Note that the generated file is "XML-identical" to the original except that, to improve readability, newlines are inserted *within* opening and closing tags - the visual appearance of the document will therefore differ from the original. DeltaXML also assigns a prefix to every element identified as belonging to a namespace, irrespective of whether it had one in the original file. This is necessary to handle merges of documents with different namespaces. You will therefore see "p0:" or a similar prefix on your element names if you were not using a prefix for one or more of the namespaces in your source file. Using XSL output filters these extra newlines and prefixes can easily be removed.

```
import com.deltaxml.api.DeltaXMLProcessingException;
import com.deltaxml.api.XMLCombiner;
import com.deltaxml.api.XMLCombinerConfigurationException;
import com.deltaxml.api.XMLCombinerFactory;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;

public class SimpleCombiner
```

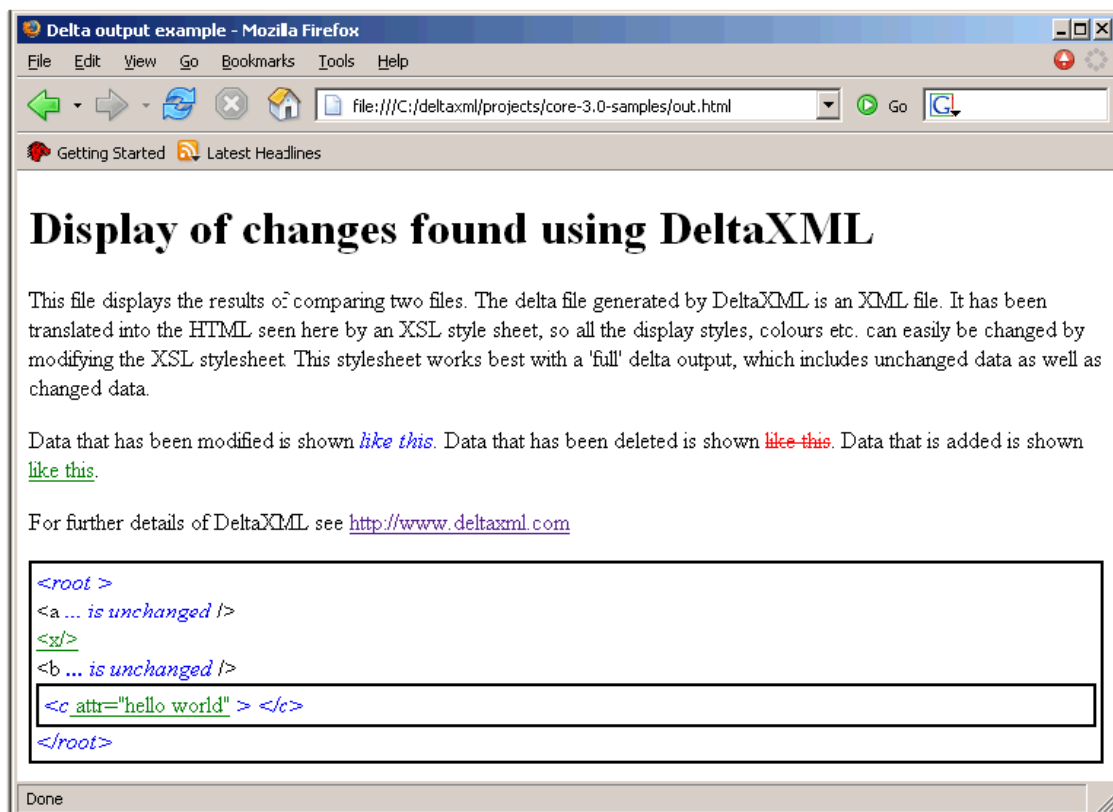
```
{
    public static void main(String[] args)
        throws DeltaXMLProcessingException,
            XMLCombinerConfigurationException,
            FileNotFoundException, IOException
    {
        XMLCombiner combiner = XMLCombinerFactory.newInstance().newXMLCombiner();
        combiner.combine(new StreamSource(new File(args[0])),
            new StreamSource(new File(args[1])),
            new StreamResult(new File(args[2])));
    }
}
```

Chapter 4. Viewing the changes

4.1. Generating XHTML with XSL

XSLT is our preferred tool for transforming a delta file into a viewable form. The `outfile.xsl` example (see the `samples` directory) demonstrates displaying a delta file as HTML using nested tables to represent the tree structure. This is an intuitive and compact display of changes suitable for viewing in a browser.

To illustrate this, the delta produced by the SimpleFiles example presented earlier can be post processed to produce the following HTML table:



To add such post processing to the SimpleFiles example, it is merely necessary to tell the `PipelinedComparator` class that it must now use an output XSL filter. This is done using the `setOutputFilters` method on the `PipelinedComparator` class. The modified class is presented below:

```
import com.deltaxml.core.PipelinedComparator;
import com.deltaxml.core.PipelinedComparatorException;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
```

```
/**
 * Demonstrates the construction of a very simple
 * pipeline doing fileIO.
 */
public class SimpleFiles {

    public static void main(String[] args)
        throws FileNotFoundException, PipelinedComparatorException
    {
        PipelinedComparator pc= new PipelinedComparator();
        List outFilters= new ArrayList();
        outFilters.add(new File("deltaxml-tables.xsl"));
        pc.setOutputFilters(outFilters);
        pc.compare(new File("a.xml"), new File("b.xml"), new File("out.html"));
    }
}
```

Note that the output file is now called "out.html".

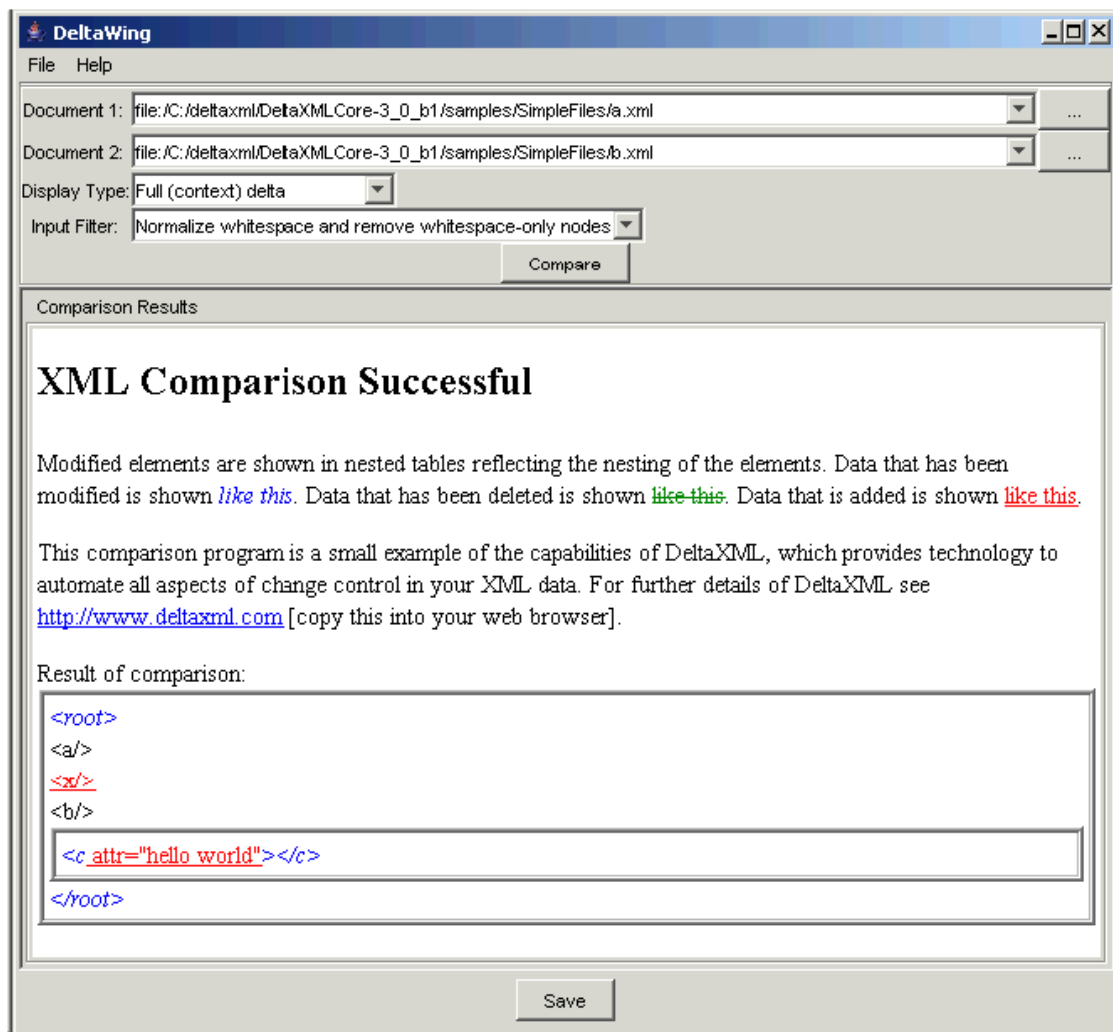
As can be seen from this example, using XSLT filters to create input and output filters to the DeltaXML Comparator within Java pipelines provides a straightforward methodology for building sophisticated display systems for users, using (X)HTML, CSS and JavaScript and requiring only standard browser technology.

4.2. DeltaWing - Building a Java UI

DeltaWing is a sample application which allows users to select two XML documents, as local files or URLs, and generates a full delta file. Using XSLT the delta is transformed into HTML which is displayed within a basic browser window within the application.

Once input documents have been selected they are fed into a TrAX pipeline (see Building a pipeline) so that all subsequent processing is in-memory. The TrAX pipeline is constructed using XSL documents which are compiled as part of the application. For this sample application the standard Swing HTML display control, which only supports HTML 3.2 and provides rudimentary functionality, has been used: For a commercial implementation, please contact us to discuss more advanced options.

The following screen dump illustrates DeltaWing being used to compare the two simple XML files.



The full source code for the DeltaWing application is included with the API distribution, which includes an Ant [<http://ant.apache.org/>] build script which allows the application to be compiled, assembled and run simply by changing to the `deltawing` directory and typing `ant`. The `deltawingex.jar` which is generated and automatically run provides a useful demonstration tool to illustrate the basics of XML comparison. To see the full delta display of the demonstration files you should select `demo1.xml` and `demo2.xml` from the `samples` subdirectory of your install dir.

Since the source for comparisons can be modified to include your own input and output filters you can use this tool to prototype transformations you require. For technical details of the application itself please refer to the source code. The (very simple) UI is constructed using NetBeans using only standard Swing components, to allow you to extend and develop it as you wish.

4.3. Advanced visual display options

Native-code display options require greater effort but provide richer functionality. More sophisticated tree views can be built - you may want to store the delta as a tree model (using `dom4j` [<http://www.dom4j.org/>] or similar) and use it to populate a `JTree` for display. For 2-pane views a change navigator may assist users by providing a visual overview of changes in the two documents, perhaps using colour to highlight changes.

For some applications "interactive" change control may be appropriate. For example, using *Next Change* and *Previous Change* buttons may give effective navigation in some circumstances: using *Accept Change* and *Reject Change* buttons it is possible to give the user fine control over merge operations.

For examples of tools providing useful UIs for viewing changes, you may want to study the ideas proposed for the following text-based differencing tools:

- WinDiff™ - see <http://www.microsoft.com> [<http://www.microsoft.com/>]
- Guiffy™ - see screenshots at <http://www.guiffy.com/shots.html> [<http://www.guiffy.com/shots.html>]
- ComponentSoftware™ - screenshots at <http://www.componentsoftware.com/products/csdiff/ScrShots/sh02.htm> [<http://www.componentsoftware.com/products/csdiff/ScrShots/sh02.htm>]
- ExamDiff™ - see <http://www.prestosoft.com/examdiff/examdiff.htm> [<http://www.prestosoft.com/examdiff/examdiff.htm>]
- Compare It™ - see <http://www.grigsoft.com/wincmp.htm> [<http://www.grigsoft.com/wincmp.htm>].

You will need choose an appropriate metaphor to handle the tree structure of XML, which is not properly handled by any of the above tools. Nesting or a "hierarchical tree view" can provide a familiar interface.

Chapter 5. Working with DeltaXML

5.1. The DeltaXML delta format

Our patented delta format is one of our core technical strengths. By representing changes in a form as close as possible to that of the original document, and in a way that is easy to process, you can benefit from pipeline architectures that permit extensive customization. And since the same structure can handle both Full Delta [coretutor-2-2.xml] and Changes Only [coretutor-2-1.xml] deltas, you can switch between a compact form and a full-featured form without altering your post-processing code.

Technical details of the DeltaXML standard can be found in our Introduction to DeltaXML [/pdf/deltaxml-intro.pdf] (PDF) and in How DeltaXML Represents Changes to XML Files [/core/deltaxml-changes-markup.html]. The format is refreshingly simple in its structure, a simplicity which pays dividends in the more complex applications of DeltaXML. Unlike an XPath-based syntax such as XUpdate [http://www.htmldb.org/xupdate/], the delta file remains human-readable, one of the key benefits of XML.

5.2. Managing white space

For many users of XML, white space is critical; for others, it is an irrelevance. For example, when working with XHTML white space can generally be ignored except within `<pre>` elements. This syntax is defined within the Schema/DTD for XHTML, and since DeltaXML (or, more properly, JAXP [http://java.sun.com/xml/jaxp/]) uses this metadata during document loading, you will get the comparison results you expect whenever the input documents are metadata-controlled.

For many documents, however, no Schema/DTD is used. For these "well-formed" documents, another approach is required. Using XSLT - input and output filters [coretutor-5-2.html] described how to preprocess input documents - by applying a standard `normalize-space.xsl` stylesheet, multiple contiguous spaces/newlines are reduced to a single space, ensuring that white space only differences will be ignored. By modifying this stylesheet it is possible to specify that only white space meeting specific criteria - maybe excluding that within `<pre>` elements - should be normalized.

5.3. Namespaces

DeltaXML offers full support for XML Namespaces [http://www.w3.org/TR/REC-xml-names/]. This includes ignoring differences between namespace prefixes when the URIs are identical, and correctly processing namespace declarations on ancestor elements. Where namespace prefixes are used within attribute values, care should be taken that they are used consistently.

The generated delta declares the DeltaXML namespace for the root element:??

```
xmlns:deltaxml="http://www.deltaxml.com/ns/well-formed-delta-v1"
```

DeltaXML also scans the input documents for namespace declarations, including prefixes, and uses these in the output delta.

On viewing a delta file, you will very probably see `p0:` namespace prefixes attached to all your elements. These are automatically assigned to elements in the default namespace and can safely be ignored, the semantics of your generated (recombined) document are identical to the original. Alternatively an output filter can be used to replace all references to this automatically assigned default namespace with references to the unnamed default namespace.

Further details of how DeltaXML handles namespaces are in the documentation for The DeltaXML

delta format [coretutor-6-4.html].

Chapter 6. Distributing your application

6.1. Requirements for distribution

The DeltaXML Core API redistributable product is contained entirely within `deltaxml.jar`. Redistribution is subject to the terms of the licence agreement. You may also redistribute our XSL scripts, either modified or unmodified, with your application, either stand-alone or "burnt -in" - for example, using the "embedded strings" approach or using Apache XSLTC [<http://xml.apache.org/xalan-j/xslt/index.html>] with `translets`.

You will also need to provide (or require) a JAXP-compliant parser and transformer. The Apache Xerces/Xalan combination supplied with the software may be redistributed under the terms of the Apache Software License [<http://xml.apache.org/dist/LICENSE.txt>].

Note that distribution of a product which is in competition with DeltaXML is not permitted, and the DeltaXML Core API and any add-ons must not be exposed in distributed products, for obvious reasons. Please check your licence and contact us if you are unsure.

Chapter 7. Troubleshooting

7.1. SAX errors - invalid XML

The most common errors when processing XML are during the initial document load - resulting from XML that is not well-formed. For these you will need to refer to the error codes for your parser - for Xerces, see <http://xml.apache.org/xerces-j/> [<http://xml.apache.org/xerces-j/>]. Commonly encountered errors are:

- XML is case-sensitive.
- Every element must be closed - for XHTML, this includes `<p>` and `
` elements
- The `encoding` must be set correctly. For Unix and Linux machines, this generally means UTF-8, for Windows machines (especially when using extended characters such as those with umlauts), ISO-8859-1 is usually a good first choice.
- Where a Schema or DTD is referenced, it must be accessible.

A validator such as that from STG [<http://www.stg.brown.edu/service/xmlvalid/>] or EICel [<http://www.elcel.com/products/xmltools.html>] can help you track down these errors quickly.

The JTidy [<http://sourceforge.net/projects/jtidy/>] project provides tools for converting poorly formatted HTML to well-formed XHTML - you may find useful code here for handling non-compliant documents.

7.2. XSL errors

Please see the error documentation for your transformer. For Xalan, this is available online at <http://xml.apache.org/xalan-j/apidocs/index.html> [<http://xml.apache.org/xalan-j/apidocs/index.html>] under the heading `XSLTErrorResources` while Saxon users can find details of error recovery policies listed at <http://saxon.sourceforge.net/saxon6.5.3/conformance.html#errorrecovery> [<http://saxon.sourceforge.net/saxon6.5.3/conformance.html#errorrecovery>].

Things to check include:

- XSL stylesheets must be well-formed XSL - see previous page
- When building a pipeline, test your XSLT in isolation before assembly.
- Examine stack traces carefully - a wrapped exception may provide error details.

7.3. DeltaXML error codes

DeltaXML in its default configuration is capable of handling *any* two well-formed XML documents which have the same root element type, and will not in these cases generate error codes. When recombining (applying a delta), especially when working with orderless data some exceptional conditions can arise - these are detailed at the online error messages page [</library/deltaxml-error-messages.html>].

Chapter 8. Further Reading

8.1. DeltaXML Documentation

The primary documentation for the API is from the Javadocs accompanying the current release - in the apidocs directory, open file `index.html` in a browser. This documentation is also available as online Javadocs [[/core/current/docs/api/](#)].

Support documentation is available from our list of User Manuals for various aspects of DeltaXML [[/library/](#)].

8.2. External Resources

XML comparison touches on many areas of activity - these are just a few of the resources we have found particularly useful/interesting:

- [xml-dev](http://lists.xml.org/archives/xml-dev/) [<http://lists.xml.org/archives/xml-dev/>] - despite the noise, most interesting developments still get a mention here
- [Zvon](http://www.zvon.org/) [<http://www.zvon.org/>] - some excellent quick tutorials
- [XML Cover Pages](http://xml.coverpages.org/) [<http://xml.coverpages.org/>] edited by Robin Cover - excellent in-depth tutorials
- [CatchXSL](http://www.xslprofiler.org/) [<http://www.xslprofiler.org/>] - a very useful XSL profiler
- [dom4j](http://www.dom4j.org/) [<http://www.dom4j.org/>] - a lightweight replacement for DOM, with an excellent architecture and XPath navigation
- [XPipe](http://xpipe.sourceforge.net/) [<http://xpipe.sourceforge.net/>] - an "XML assembly line" architecture
- [RELAX NG](http://www.oasis-open.org/committees/relax-ng/) [<http://www.oasis-open.org/committees/relax-ng/>] - a well-designed meta-language that is easier to get to grips with than W3C Schemas
- [Apache XML Project \(again!\)](http://xml.apache.org/) [<http://xml.apache.org/>] - resources that have helped speed the world-wide adoption of XML
- [W3C](http://w3.org/) [<http://w3.org/>] - home of XML

8.3. Useful Books

Of the many books devoted to XML we have found the following particularly helpful.

- [Processing XML with Java](http://www.ibiblio.org/xml/books/xmljava/) [<http://www.ibiblio.org/xml/books/xmljava/>] by Elliotte Rusty Harold (online)
- [XSLT](http://www.oreilly.com/catalog/xslt/) [<http://www.oreilly.com/catalog/xslt/>] by Doug Tidwell (O'Reilly)
- [XSLT Programmer's Reference 2nd Edition](http://www.wrox.com/books/0764543814.shtml) [<http://www.wrox.com/books/0764543814.shtml>] by Michael Kay
- [XSLT and XPath on the Edge](http://www.jenitennison.com/xslt/index.html) [<http://www.jenitennison.com/xslt/index.html>] by Jeni Tennison.