

A Gentle Introduction to XSL-FO and XEP

Table of Contents

1. Is this for you?	1
2. An overview of the process	1
3. Tools you will need.	2
4. Using the XEP Assistant to produce PDF output	3
5. Using the command line	3
6. Error recovery. When it goes wrong	4
7. XEP in other environments	6
Glossary	7

1. Is this for you?

This document is for the first time user of XEP 4.9. If you have only limited experience with [XSL-FO](#), or are unfamiliar with the process of using XSLT and a formatter such as XEP, then this document is for you.

Working from a simple (pre-prepared) XML example, you will produce both [PDF](#) and [PostScript](#) output using XEP 4.9

This document shows two ways of working with XEP, one graphical and one using the command line.

Later, we can introduce errors into the example, and you will see how to find and correct them.

The XML document and its associated XSLT stylesheet to make a file ready for the XEP formatter are available in the `~/examples` directory of the XEP installation. I'm using the tilde (`~`) as a replacement for the directory into which you installed XEP 4.9

Once you are familiar with the transformation, further options are offered to show ways in which you can control the operation of XEP to generate the output you want.

After reading it you will be able to use XEP with your own documents and the other components of XEP documentation as a reference.

2. An overview of the process

XEP is a formatter. A tool used as part of a chain normally used to transform an XML input document into a printable document for distribution. The process consists of two steps. The first transforms the input document into a standard XML format (XSL-FO), the second generates the printable format from that intermediate form. XSL-FO can be considered as a specification for the formatting that you want for a particular document type.

Starting with the file `hammer.xml`, a small XML file which is provided with the distribution of XEP 4.9. If you installed XEP into `\xep` (or `/xep` on *nix), then this file will be found in the `/xep/examples/hammer` directory. If you are using a Windows operating system, please change all such path information from using the forward slash (`/`), to use the (`\`).

Open the file in any text editor. You will see its structure, having a root element named `manual`, containing the remainder of the document, it has `chapter` elements, each of which contains paragraphs (in `p` tags) and so on. Use this file for your experiments until you have your own material and the knowledge to develop your own stylesheets. If you want to change it, keep a copy of the original file so you know what worked.

In the same directory is a file named `hammer.xsl`, which is an XSLT stylesheet used to convert the source XML document into a specification for the XEP formatter. Feeding this specification to XEP can produce a PDF output document ready to print.

This is the end to end process which will be described.

The input file, marked up in XML, is fed to an XSLT transformer. XEP comes with one provided. The job of the transformer is to take the content in the source document, and depending on the element name and position (e.g. a paragraph or a list or a chapter title), to modify (transform) the markup or content, into the form used by XSL-FO, which specifies how XEP 4.9 should present the file ready for viewing or printing. If you want to see this vocabulary in use, have a look in the `examples` directory where you will find files with an extension of `.fo`, which is often used for this type of file.

The second step feeds the XSL-FO document to the formatter, which creates and fills pages according to the specification laid out in the XSL-FO document.

Finally, dependant on the format specified, the formatter writes out the pages in either PDF or PostScript.

Since most users simply want to produce PDF or PostScript from XML (or XSL-FO) XEP 4.9 may be used to perform all these processes as a single step.

3. Tools you will need.

I'll assume you have XEP 4.9 installed on your computer.

In order to view Adobe PDF (Portable Document Format) files, you will need a viewer. Adobe provide a free one.

If you don't have it on your computer, you can download and install it from the [Adobe website](#), free for personal use. Note where you install it, since you can set this in XEP Assistant, and view the PDF output file immediately when XEP has processed it.

To view PostScript files, one of the options is to use a product called [GhostView](#). Versions are available for most operating systems. This product is also capable of viewing PDF.

4. Using the XEP Assistant to produce PDF output

With XEP is distributed a tool which enables the easy transformation of files in a graphical environment, suitable for users more familiar with a graphical interface.

Find the `x4u.bat` file in the distribution and click on it. This runs the Assistant. Under the file menu, select the `hammer.xml` file as the input file to process. Under the Formatting menu, select start, then tick the apply stylesheet box, and select `hammer.xsl` from the same directory. Since this is an XML file, both steps of the process are needed, the transformation to XSL-FO format, and the formatting into PDF.

The lower part of the screen may be used to select a viewer, either Adobe Acrobat *reader* or *Ghostview*. The *Assistant* needs to be told where you installed the application. Use the browse button to locate the application. Tick the **display with** box if you want to launch the viewer once the process is complete.

Either remember the value in the output file selection box, or change its value to indicate where you want the output to be located. The default output is set as PDF. Leave it as that, then select OK.

The right hand tab marked event log will show the file progress through to completion.

The output file may be read with the Adobe Acrobat reader.

Caution. Updating PDF files whilst viewing them.

The Acrobat reader will not allow a file to be overwritten whilst it is open. If you wish to modify the source XML file, and re-process it to produce PDF, you must close the file in Acrobat reader, or XEP will not be able to re-write the modified file. This applies whichever route you take to producing PDF.

5. Using the command line

For those used to the flexibility of the command line interface, one is provided with XEP.

If you've never used the command line on Windows, it is available from the start menu, programs, and either directly as an option or as a tab from the accessories tab. Normally called **Command prompt**. The window that pops up is the command line interface to Windows. If you type **help** it responds by listing all the commands it understands. You may want to create a temporary directory to work in, perhaps `c:\myfiles`. If you avoid using spaces in the directory names, it makes working with the command line that much easier.

Taking the same example as previously, to format `hammer.xml` using `hammer.xsl`, to produce `hammer.pdf`, the following command line is used. The script **xep.bat** is a part of the distribution, found in the top level installation directory.

The command below is split over 3 lines for readability. It should remain on one line, so just remove the `\` characters used to indicate a continuation line.

```
>\xep\xep.bat -xml \xep\examples\hammer\hammer.xml \
               -xsl \xep\examples\hammer\hammer.xsl \
               -out hammer.pdf
```

Finally, as a command line exercise, to produce `hammer.fo`, the matching XSL-FO file. If you can find out where `java` is installed, and add it to your environment path variable. If you installed XEP into `/xep`, the command is

```
>java -jar /xep4/lib/saxon.jar -o hammer.fo \
      /xep4/examples/hammer/hammer.xml \
      /xep4/examples/hammer/hammer.xsl
```

This uses the *Saxon* XSLT engine, another *java* tool, to produce `hammer.fo`, in the current working directory. *Saxon* takes its parameters as shown. The output is specified using the `-o` option, then the input file and stylesheet are appended. So the general form is

```
>java -jar saxon.jar \
      -o outputFile \
      inputFile \
      stylesheet
```

If that makes sense to you, you can now start to use your own XML files to produce output suitable for the XEP formatter.

6. Error recovery. When it goes wrong

Rather fake, but in order to find and correct errors it is necessary to see how they are reported. Two potential sources of errors are possible. Firstly in the source XML document. It could be either syntax or validity. Secondly, an error in the input to XEP. Often generated by an XSLT conversion, where the properties are mis-used. XEP has a tool to validate this input and isolate the source of the error.

Firstly then the source code error. Open the `hammer.xml` file, near the top there is an annotation element. Modify the markup as shown below

```
<annotation>
  <p>
    The purpose of this document is to illustrate handling complex
    layout patterns in the Extended Stylesheet Language (XSL).
    To simulate a real text, I have written an operation instruction
    for a hammer. The genre of operation instructions turned out
    to be quite convenient for showing most standard formatting
    properties. Features used in this text include:
  </b>
```

The closing `p` tag has been replaced by a `b` tag. The structure is now unbalanced, in a manner that is called not well-formed. Run this through either *Assistant*, or the command line *XEP*. *Assistant* produces the following report.

```
[error] javax.xml.transform.TransformerException: \
org.xml.sax.SAXParseException: \
unexpected characters in element end tag (expected "p")
```

Again the lines are split using the \ character, where on the screen they are on a single line.

The message is clear. The software expected an end tag of p, and didn't find one. No indication of where though. For a long document this is a problem. For shorter ones perhaps your XML editor of choice can help you.

If you have no other help, then use the tools provided with XEP. Open up an MSDOS window (Command prompt), or a shell, and selecting the correct paths, type in the following, repeating what the use in [Using the command line](#) section above.

```
>java -jar \xep\lib\saxon.jar -o hammer.fo hammer.xml hammer.xsl
```

This time the error reporting is clear and concise

```
>java -jar \xep4\lib\saxon.jar -o hammer.fo hammer.xml hammer.xsl \
Error on line 23 column 7 of
  file: .. /hammer.xml: \
  Error reported by XML parser: \
  unexpected characters in element end tag (expected "p")
Transformation failed: Run-time errors were reported
```

Your display will vary a little from this, dependant on the location of your files, but the information is clear. Somewhere on line 23, the **end tag** is suspect. Now remove the error, by replacing the `</b>` with a `</p>`, and re-run the transformation. The error will not show if you have cleared it.

This demonstrates the increased detail available from a two step process, which is just a nuisance on a correct file.

If you ran the command line transformation, using saxon, then you will have a file named `hammer . fo`, into which we can now insert an error, to view the second source of errors.

In summary. When converting from XML to PDF (or PostScript), errors may be detected in either of the two stages of the transformation. It is often easier to isolate from which stage the error is originating. Now to inject and detect an error in the second stage, the XSL-FO to PDF stage.

Open up the `hammer . fo` file, using any text editor. Far more difficult to read, since much more is happening, but you should see the structure of the file if you spend a few minutes with it. In particular, look for that first paragraph again, where we inserted the error before. It looks different, since its now ready for the formatter. Look for

```
<fo:block space-before.optimum="6pt" text-align='centre'>
  The purpose of this document is to illustrate handling complex
  layout patterns in the Extended Stylesheet Language (XSL).
  To simulate a real text, I have written an operation instruction
  for a hammer. The genre of operation instructions turned out
```

```
to be quite convenient for showing most standard formatting
properties. Features used in this text include:
</fo:block>
```

An error has been introduced, specifying that the text should be aligned **centre**, which is a misspelling according to the official recommendation, which uses the American spelling (center). A perfectly innocent typographical error. Now to find it.

Note: If you regenerate the pdf file, this intermediate XSL-FO file will be overwritten, as mentioned above.

If you introduce the error as shown, then run the command shown below, the *validate* program should find it.

```
>\xep\validate hammer.fo
file: ... hammer.fo: line 2
Attribute 'text-align' cannot have a value of "centre".

hammer.fo: 1 error
```

The *validate* script, located in the distributeion directory has a single purpose, to spot these errors. The output could not be clearer, the text-align property cannot take a value of centre. To remove the error, either change the value to **center**, or remove the attribute and its value.

Re-run the validator, and a successful validation will result. Since the normal source of such errors is the XSLT stylesheet, that is where you would normally locate and remove these errors, since the purpose of the stylesheet is to convert the source XML into the form you have just seen in hammer . fo.

That is a view of finding errors in the two stages. Although not normally done in two separate stages like that, it is worth remembering that it can be done this way, since it is often easier to locate errors, and the objective is to produce the output efficiently.

7. XEP in other environments

Since this document is only an introduction, no detail is given of the full suite of options. That is available in [the reference](#). It is worth knowing of other applications of XEP 4.9

The [RenderX](#) website has a page devoted to alternative ways of configuring XEP 4.9 It is worthwhile appreciating these, they may be in use now, or in the future, in your organisation.

Be aware that XEP 4.9 may be readily integrated into all these products.

The jEdit plugin

jEdit is a cross platform editor with an amazing range of plugin components.

The oXygen editor

The oXygen xml editor provides built-in support for XEP. RenderX offers a bundled distribution, Docbench, in which XEP and oXygen are tuned to work with each other.

As an Ant task

Ant is a java based building tool. XEP distribution includes a Task for Ant.

Cocoon

A web development framework

As part of a Java servlet

For Tomcat server users.

As an Enterprise Java Bean

XEP can server as a remote or local EJB in J2EE environments.

Glossary

PDF (Portable Document Format)

PDF is a universal file format that preserves the fonts, images, graphics, and layout of any source document, regardless of the application and platform used to create it. See [Adobe](#) website for more information.

PostScript

Adobe® PostScript® is the worldwide printing and imaging standard. Used by print service providers, publishers, corporations, and government agencies around the globe, Adobe PostScript 3 gives you the power to print visually rich documents reliably. See [Adobe](#) website for more.

XSL-FO

XSL, Formatting objects. A standard way of specifying how content should be presented. A World Wide Web Consortium specification. [W3C](#)