Hands-on XSL
XSL for fun and diversion

Don R. Day
Advisory Software Engineer, IBM
March 2000

This article presents a simple, hands-on exercise that demonstrates the principles of the Extensible Stylesheet Language Transformations (XSLT). It takes about an hour to complete the concept exercises and about 15 minutes at a computer to try out the results with a real XSLT processor.

Contents:

XSL is a style sheet language for documents marked up using XML, the Extensible Markup Language, which can be thought of as "SGML for the Web." XSLT is used to describe how an XML source document is transformed into another XML document that uses the XSL formatting vocabulary. However, XSLT may be used for other general transforms as well. This exercise will deal with one of the most practical tasks for XSLT, transforming an XML document into an HTML document that can be viewed by any Web browser.

For your information, this exercise demonstrates section 5.1 of the "XSLT Recommendation, Processing Model" (see Resources). Don't expect to understand this paragraph right now, but come back after completing the exercise to see if this quote from the spec makes more sense.

> "A list of source nodes is processed to create a result tree fragment. The result tree is constructed by processing a list containing just the root node. A list of source nodes is processed by appending the result tree structure created by processing each of the members of the list in order. A node is processed by finding all the template rules with patterns that match the node, and choosing the best amongst them; the chosen rule's template is then instantiated with the node as the current node and with the list of source nodes as the current node list. A template typically contains instructions that select an additional list of source nodes for processing. The process of matching, instantiation and selection is continued recursively until no new source nodes are selected for processing."

The key features of XSLT are:

- It is recursive.
- It operates on an XML source document that has been parsed into a source tree.
- It specifies the transformation of the source tree into a result tree.
- And finally, it copies the result tree into a result file in a specified format.

This paper exercise demonstrates each of these features.

Throughout this exercise, Glossary terms link to their definitions.

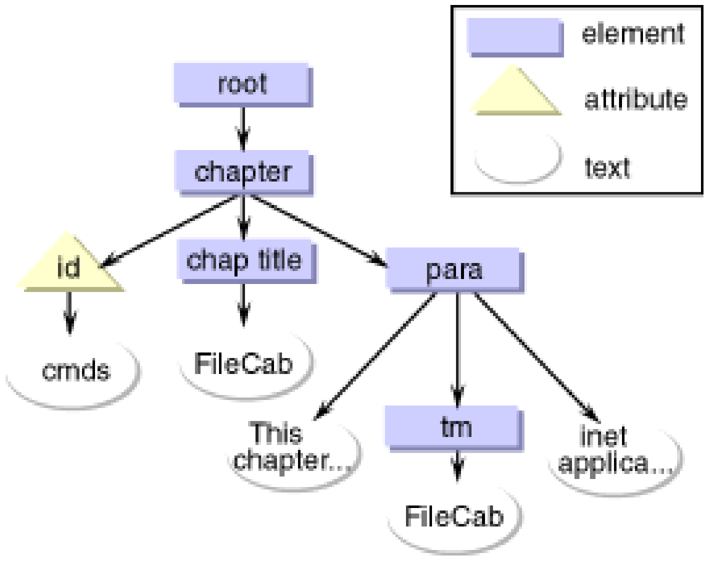
Sample XML document

This is the sample XML document that we will use for this exercise:

```
<chapter id="cmds">
 <chaptitle>FileCab</chaptitle>
 <para>This chapter describes the commands that manage
 the <tm>FileCab</tm>inet application.</para>
 </chapter>
```

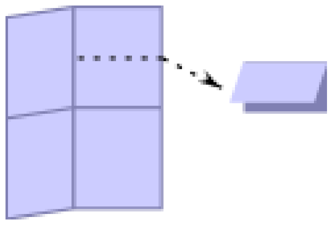
Note that the first element, `<chapter>`, surrounds the entire document. Therefore, it is the document element of this document instance. It is also the child of a root node that is defined to always be the top node of any parsed XML document.

Concept: Source document viewed as a tree
This is how the source XML document may be represented after it has been parsed, as the first stage in XSL processing.

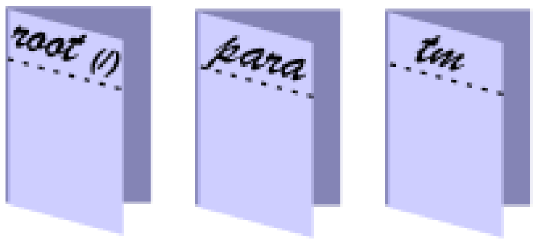


For another way of viewing the parsed source tree for this document, see the definition for walking a tree.

Preparation for the exercise
Fold and tear two sheets of colored paper (8 1/2" x 11") into quarters. Then fold each quarter in half again, to produce eight small leaflets.

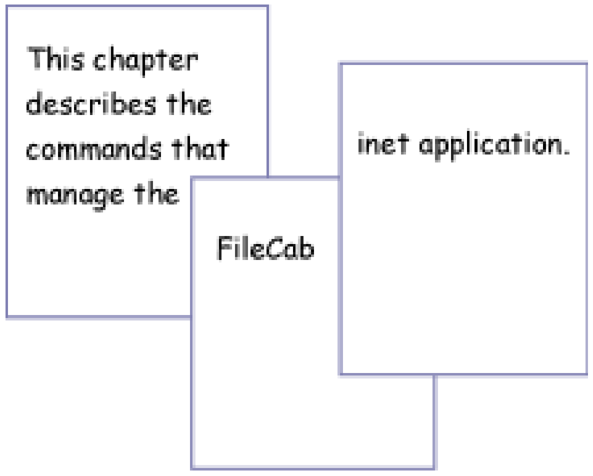


Draw a line across the top of the front of each leaflet. Above the line, label the leaflet with the name of each node (element or attribute) that shows in the source tree view of the sample document (refer to Concept: Source document viewed as a tree). These labeled leaflets represent the template rules that make up an XSL style sheet.



and so on.

Fold and tear one sheet of white typing paper into eighths. (This will be more than enough pieces for the exercise.) On one side of each piece, write the text portion that corresponds to each element or attribute. These pages represent the text or character content of a source document. Here is how you would prepare the text pages for the content of the paragraph node, according to the parse tree view (refer to the Concept: Source document viewed as a tree illustration):



Drop the folded leaflets (not the white text pages) into a small box, bowl, or sack. Altogether, this represents a complete XSL style sheet. If you want to carry the analogy to its fullest, label the front of the container with this text:

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/TR/xhtml1/strict">
```

Label the backside with:

```
</xsl:stylesheet>
```

Concept: Namespaces
In this example, `xmlns` denotes a namespace, or a qualifier for elements introduced from other XML vocabularies. The URL is a pointer to more detail about the kind of thing on the right side of the colon (:). So the `xmlns:xsl` line indicates the URL where there is defining information about markup that starts with `xsl:`. The next line, with `xmlns` by itself, points to more information about the default markup in the style sheet, which will be valid XHTML.

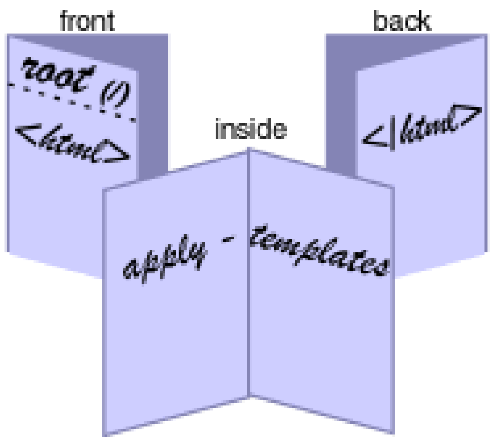
Define the templates
Each leaflet made from the colored paper represents a template rule that defines the desired processing for each node (usually elements) in the source tree. The outer front and back faces of this leaflet represent the start and end tags of the element.

The leaflet itself is a container for the templates that match child elements, as in the <chaptitle> and <para> children of the <chapter> element. This represents the recursive aspect of XSL processing: starting from the root node of a source tree, each node is inspected to see if there is an applicable template rule; if so, that rule is invoked and then the inspection continues on the children of the new node. Whenever you intend to process the child nodes of an element, you are applying the templates that exist for those elements. Hence, this recursive action is called *apply-templates*.

If you want to skip the processing of an element's child nodes or its content, you simply leave out the call to apply-templates. For example, there is not much point to having such a call for the template of an empty element -- it has no child element nodes. Similarly, if an element contains an attribute that indicates it has confidential information, you may filter out that content by simply not referencing it with the apply-templates call.

Refer to the Sample XML document as you follow this exercise to create a result tree. As an extra measure of keeping the result tree distinct from the source tree, write all the markup for the result tree in upper case (for example, using <B> instead of <b>).

Step 1
Start with the template for the root node, also designated by a slash (/). The template rule for the root node is always processed first, if it is present. Because your goal is to turn the sample XML document into an HTML article that can be viewed in a Web browser, this particular template rule is a great place to generate the start and end markup for the desired HTML result, as well as any comments or metadata that apply to the overall result. On the front page of this leaflet, below the label, write "<HTML>" and on the backside write "</HTML>". Because we want to process the rest of the document recursively, write "apply-templates" across the middle of the inside. This now represents a complete template rule for the root node of the document. Your template rule might look like this, front, inside, and back:



Step 2
Next, for the `<chapter>` element, write the following on the front of its leaflet, beneath the "chapter" label:

```
<HEAD>
<TITLE>My result doc</TITLE>
</HEAD>
<BODY>
```

Because we want to process the rest of the document recursively, again write "apply-templates" across the middle of the inside. On the backside write:

```
</BODY>
```

This completes the second template rule.

Step 3
Continue in the same way for the rest of the element nodes that are in the source document viewed as a tree diagram. Be sure that inside each leaflet or template rule you write the phrase, "apply-templates."

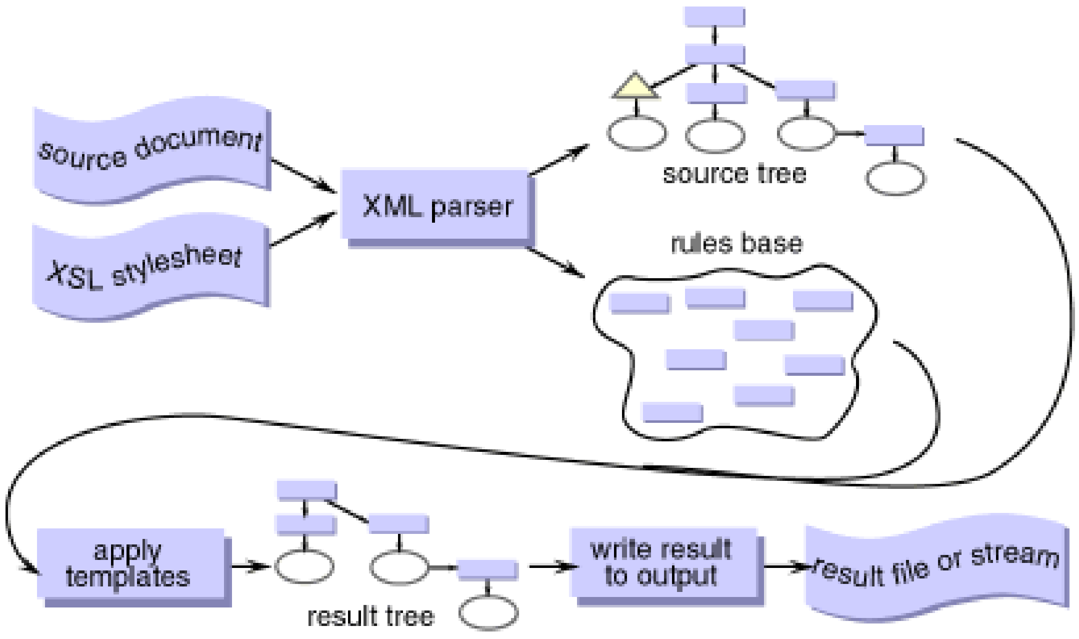
Here are some suggested things to do with the rest of the template rules:

- **id:** If you have started a template rule for this item, put it aside for now. Not everything in a source document needs to be transformed into something in the resulting document. Attributes usually represent a property rather than a structure, and for the sake of this lesson, we are dealing only with one-to-one structural transformations.
- **chaptitle:** Turn this into an H1 heading followed by a horizontal rule (use this XHTML-compliant format for now: `<HR/>`).
- **para:** Turn this into a well-formed HTML paragraph (`<P>` and `</P>`).
- **tm:** Highlight the content of this element by wrapping it with HTML's bold phrase markup (`<B>` and `</B>`). For extra credit, add `<SUP>(TM)</SUP>` on the back page of the leaflet. This demonstrates that a template can be used to insert generated text adjacent to an element's content.

After you have defined a template rule for each element node of the source tree, gather the leaflets and drop them into your style sheet container (box or bag). Stir them around -- this reinforces that there is no particular sequence for rules in a style sheet, other than however you might organize them for the convenience of editing later on.

We are now ready to begin our transformation on the source document.

Concept: The XSL processing sequence
An XML parser converts a source document into a source tree, then it reads in the XSL style sheet and organizes the template rules for efficient lookup. Then the XSL processor "walks" the source tree starting from the root node, and attempts to match each node to a corresponding template rule. If such a match is made, the template is copied into the result tree, and processing continues until the source tree has been completely traversed. At the end, the XSL processor walks the result tree and copies what it finds into an output file or stream, using the syntax implied by the `xmlns` attribute on the xsl:stylesheet container.



Assemble the result tree view
You will now do an exercise that mimics the XSL processing sequence.

The picture in Concept: Source document viewed as a tree represents the XML source document that has been parsed into an internal structure of nodes and properties.

The XSL processor retrieves, parses, and organizes the template rules in the style sheet. Take the template rules leaflets out of their container and lay them out so you can see their labels; this represents creating the rules base part of the previous diagram.

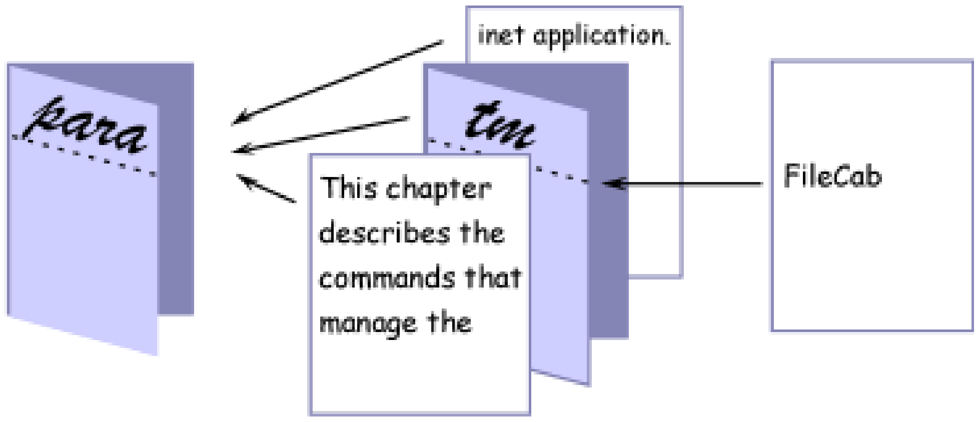
Next, you will model the apply-templates part of the process.

To begin applying the templates, locate and pick up the template rule for the root node. Open it and note that inside is the instruction apply-templates. This means you must go to the next node of the tree, `chapter`, and see if there is a template rule that matches it. Did you find it? Okay, pick up that leaflet and tuck it inside the first leaflet. This represents that we are recursively building a result tree based on the structure of the source tree.

For each node in the source tree, keep doing the same thing. Note that there is an apply-templates instruction for the `chapter` template rule, so go to the next node in sequence, `chaptitle`. Find the leaflet for that template rule and tuck it inside the previous template, then check its inside. Again, it says that we are to apply-templates, so get the next child of `chaptitle`, which in this case will be the white text leaf that says "FileCab." Tuck this page inside the `chaptitle` template.

According to the source tree illustration, you should be at the end of that first branch of the source tree. There are no more children to find, and no more rules that apply in this location. Therefore, follow the tree back up to the next sibling element of `chaptitle`, which is the `para` element. Find its template, and continue this same process.

Processing the `para` template rule should result in this particular assembly of templates and text, corresponding to the source tree branch for the `para` node.



When you are finished, you should be holding a single, thick `root` template that contains the result tree based on the source document and the transformations that you indicated in your style sheet's template rules.

Feel free to disassemble your booklet and repeat the steps to assemble the result tree view a few times. These steps are the main point of this tutorial and should become imprinted in your mind.

Generate your result HTML
Leaf through your assembled result tree page by page, like reading a book. Write down on a separate sheet of paper everything you had written, in sequence, on the front and back colored pages (except the labels). Be sure to check the back pages of each colored leaflet for any end tag results you may have specified. As you come across them in sequence, copy the contents of the white text pages.

Voilà! You have just completed an XSL transformation, exactly the way it happens for real. No way, you say? See what's next!

Convert your paper template rules into an equivalent XSL style sheet
Now you will transfer these paper templates into an actual XSL style sheet. Get out a clean sheet of paper to write on, or turn over the page with your HTML practice output.

Separate into a pile the colored, folded pieces of paper that you have labeled. Each of these is a template rule that has a match pattern and a corresponding template for the resulting transformation.

Start writing your XSL style sheet by copying down the label you put on your container for the templates:

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/xhtml1/strict">
```

Now you will do the following for each colored leaflet in front of you. Start with the template rule for the root node, and repeat the process for the rest of the leaflets.

1. Pick up a leaflet. Use the following text as the template for what you write on the style sheet. Words that are bracketed by underscore characters represent things on your template rule that you will copy into this new format.

   ```
   <xsl:template match="__label__">
     __first_page_content__
       <xsl:apply-templates/>
     __last_page_content__
   </xsl:template>
   ```

2. Copy the leaflet's label into the designated `__label__` field of the match attribute. (Hint for the root template: instead of the name *root*, XSL actually uses the shorthand "/" (just as in a filepath) to indicate the root.)
3. Copy the first page content into its designated `__first_page_content__` location in the template.
4. Copy any last page content into its `__last_page_content__` area.

   Note that if a template had omitted an apply-templates call, you would not transcribe the `<xsl:apply-templates/>` markup shown above.
5. Do the same for each of the rest of the colored leaflets before you. Remember that the white pages were part of the source document, not part of the transformation rules. Only transcribe the information you put on the colored, folded leaflets.

Conclude the style sheet with the closing markup from the container:

```
</xsl:stylesheet>
```

Congratulations on having written a conforming, functional XSL style sheet!

Try it out
If you want to try this as an actual XSL application, do the following on your PC. Any flat editor, such as Notepad on Windows, is fine for creating the first two files.

1. Copy the sample source document into a file called `test.xml`. Save it in an obviously-named directory (for example, `c:\testxml`).
2. Copy your style sheet into a file called `myview.xsl`. Save it in the same directory.
3. Download any of the popular implementations of XSL transformations. If using Windows, XT is a readily available choice. Go to James Clark's Web site (see Resources) and find the link, "XT packaged as a Win32 executable." Open a

DOS window and unzip this file into a directory that is in your path statement. (If you opt to use a different processor, such as Saxon or Xalan, follow the setup and invocation instructions provided with those tools.)

4. Change to the directory where you saved your sample files (for example, `cd testxml`), and run this command (specific to XT) to test your style sheet:

   ```
   c:\testxsl>xt test.xml myview.xsl myresult.htm
   ```

   After a moment, the process should finish and the directory should contain the expected result file, `myresult.htm`.
5. The last proof, of course, is to open up a browser and open this file to see how it looks.

Summary
You have now completed the hands-on exercise. With your newly acquired understanding of the XSL processing model, reread the quote from the "XSLT Recommendation" that appeared at the beginning of this exercise. This description of the processing model should make more sense to you, now that you've experienced XSL, "hands-on!"

Resources

- This exercise demonstrates section 5.1 of the XSLT Recommendation, *Processing Model*.
- To try out an actual XSL application, download XT, an implementation in Java of XSL Transformations, from James Clark's Web site.
- For information about node relationships, see XPath on the W3C Web site.

Glossary
Terms and phrases used throughout this article are defined here.

attribute. A property of an element. Attributes are often used to attach such information as style, security level, intent or role, or an indication of data type.

child. A node within the scope of another node.

current node. The node that has been matched by a particular template rule. The context around the current node includes itself, its ancestors (parent, grandparent, and so on), siblings, and children.

document element. The outermost element of an XML document or fragment.

element. The structural part of an XML document. Elements have a start tag, content, and an end tag. The start tag of an element may have additional properties, called attributes, that apply to the entire element.

generated text. Data that is inserted into a transformed node of a document. In books with chapters, the word *Chapter* is usually generated as part of the printed chapter title.

match pattern. The name of an element, an attribute, or other valid XPath target in a source document.

node. An element, attribute, or text item in a document. These represent parts of the structure that have nested relationships with each other, and this nesting may be represented graphically as nodes in a tree view of the structure.

parent. The node that contains the current node.

parsing. The processing that separates the markup of a document (including tag names and the syntax, or "<" and ">" characters) from its content (usually text and images).

recursion. In a document tree structure, applying a process to each node of the tree by repeatedly calling the same process until all children and grandchildren of the node have been processed. The alternative, and the way most people think about documents, is sequential processing.

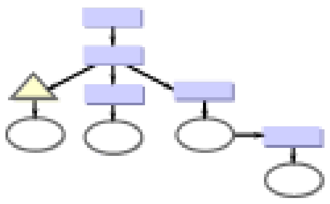
result tree. The representation of a document after it has been transformed from its source structure. The syntax for the desired result format is applied when the result tree is written to its file format by the XSL processing application.

root element. The implicit highest-level node of a parsed XML document. You may not always be able to predict which element will be the document element of a parsed instance, but it will always have a root node that you can count on being able to use for preliminary or setup processing.

root node. The starting point of all parsed documents. After the root node comes the document element, which may be different between any two documents. Because all parsed documents have a root node, this is a reliable node to use in a transform.

sibling. One of several nodes within the scope of another node.

source tree. The representation of a document after it has been parsed. Any XML syntax (the "<" and ">" delimiters) has been removed. The tree structure of a document is often diagrammed like this:



stream. Data that is in the process of being transmitted from one operation to the next. An example of streaming media is an audio file that is played on a Web browser even as the data is coming from a server.

template. The replacement markup or content to be copied into the result tree.

template rule. A style sheet is composed of a set of template rules. Each template rule has a match pattern that associates it to an element or node in the source document. The template rule contains the template of either markup or generated text that is to be associated with the result instance of the rule. A template may optionally include either a recursive call for the processing of child nodes of this node (<xsl:apply-templates/>), a call to a different, named template rule, or a query for specific values of other nodes in the source tree.

walking a tree. To travel downward along the left side of all nodes until you reach the end of a branch, then come up to the next new branch, work it to the end, back to the next branch, and so on. Walking this tree will result in this sequence of nodes visited:

root:
. . **chapter:**
. . . . id: "cmds"
. . . . **chaptitle:** "FileCab"
. . . . **para:**
. . . . . . "This chapter..."
. . . . . . **tm:** "FileCab"
. . . . . . "inet applica..."

well-formed. An XML document (sometimes called an *instance*) that meets certain criteria. For example, the document always has start- and end-tag markup for each document element (as in `<b>Hello</b>`). Elements that have no required content may be represented by an empty start tag having a slash at the end of the tag to indicate that it is empty (as in `<hr/>` for horizontal rule in HTML). Also, elements are properly nested (as in `<b><i>Hi</i></b>`), not overlapping (as in `<b><i>Hi</b></i>`). Such a document is *well-formed* and ready for use by any XML browser or processing system.

About the author
Don Day is an advisory software engineer for IBM. For the past 15 years, he has designed and supported publishing tools for IBM's Information Development community. Don provides XML expertise for Information Design and Development in IBM's e-business Operating Systems Solutions area and for IBM Corporate User Technology. He has represented IBM on the W3C XSL Working Group and is presently IBM's alternate rep for the W3C CSS Working Group. Don holds a dual-major Bachelor of Arts degree in English and Journalism and a Master of Arts degree in Technical and Professional Communication (with a minor in Computer Science) from New Mexico State University. You can contact Don at dond@us.ibm.com.