

Connect

Information Technology at NYU


[Search This Site](#)

F a l l 2 0 0 3 E d i t i o n

Browse the table of contents, or select an option from this menu:


[Current Issue](#)

[Archives](#)

[About Connect](#)

Humanities Computing

[Print this article \(315K PDF\)](#)

Publishing XML Files on the Web

By **Matthew Zimmerman**

The Humanities Computing Group within ITS' Academic Computing Services is working with the FAS Department of History on a web-based publication project, *The Public Writings of Margaret Sanger*. This online edition is part of a much larger endeavor, *The Margaret Sanger Papers Project*, which includes an already completed microfilm edition of over 9,000 of Sanger's documents, plus a four-volume book edition of Sanger's papers, the first volume of which has been published with the title *The Woman Rebel, 1900-1928*.

Although, when completed, the online edition will be much smaller than either the microfilm or print edition (600 documents), it represents an important part of the project, as it will create a new level of accessibility to the writings of Sanger, a noted social reformer and birth control advocate in the first half of the 20th century.

Project Director and Editor Dr. Esther Katz, Assistant Director and Associate Editor Cathy Moran Hajo, and Associate Editor Peter C. Engelman have chosen XML as the format in which to store the transcribed Sanger documents. XML was selected because it is the standard storage method for transcribed documents in the field of digital libraries and archives. One reason XML is the standard is its application- and platform-independence, which allows for interoperable storage, exchange, and usage. Since XML files are text files, they can be stored in the file system of any operating system (Windows, UNIX, Linux, Macintosh) and be read and manipulated by any program (word processor, web browser, Java program, Perl script, PHP, etc.).

XML and HTML

Another reason XML is the standard is its allowance for rich, "descriptive" markup. Although HTML can be used to describe the structure of a document, it is typically used to indicate how a document should be displayed, and is limited to demarcation

of headers, paragraphs, line breaks, divisions, and tables, plus some "presentational" information such as font style and size. With XML, on the other hand, it is possible to indicate that strings of text are names, dates, places, titles, footnotes, and so on, or to include whatever categorization is desired.

The code below illustrates the differences between HTML and XML. * The first section shows how a document would typically be marked up in HTML:

```
I think the City of <b>Philadelphia</b> owes a vote of thanks to this Committee for bringing together this brilliant group of speakers to discuss the subject as it has been discussed today. After listening to some of the papers that I heard, I was reminded of something <i>H. G. Wells</i> said last year in talking about the advance of culture and intelligence. He said ["I have one test of intelligence and that is the attitude of the man or woman on birth control, as that is the real test of intelligence,[" and I think if you heard, as I did, the statements made today and if you compared those statements and the attitude of mind to the same thoughts that were expressed at this sort of a conference years ago, if we would put these thoughts into the hands of the legislators it would be a great help and the opposition would be forced to come to some sort of a conclusion and let us hear what they have to say.
```

And this is the same text marked up in XML:

```
I think the City of <place reg="Philadelphia, PA" rend="bold"> Philadelphia</place> owes a vote of thanks to this Committee for bringing together this brilliant group of speakers to discuss the subject as it has been discussed today. After listening to some of the papers that I heard, I was reminded of something <person reg="Herbert George Wells" rend="italics">H. G. Wells</person> said last year in talking about the advance of culture and intelligence. He said "<quote who="WELHE">I have one test of intelligence and that is the attitude of the man or woman on birth control, as that is the real test of intelligence</quote>," and I think if you heard, as I did, the statements made today and if you compared those statements and the attitude of mind to the same thoughts that were expressed at this sort of a conference years ago, if we would put these thoughts into the hands of the legislators it would be a great help and the opposition would be forced to come to some sort of a conclusion and let us hear what they have to say.
```

Although this is a rudimentary example, it illustrates how XML turns markup from a tool for describing how a document should be displayed into a system for storing data. If we take these two snippets of code as examples, a program could be written to search the XML document for any person mentioned in the text by looking for the <person> element. Then an index of people mentioned in the text could be created. This would not be possible with HTML.

The irony of XML is that one of its greatest strengths, platform- and application-independence, is also one of its weaknesses. Since it is a relatively new technology and since it is not tied to any one program or system, XML can be difficult to work with. The publication of XML files on the Web illustrates this difficulty. Unlike HTML files, which display very nicely across a range of web browsers, documents marked up in XML alone can produce variable and often unsatisfactory results when viewed in a browser.



Figure 1. A sample XML document when viewed directly in Mozilla on Macintosh OS X.

Figure 1 shows one of our Sanger XML documents as it appears when viewed directly in Mozilla on Macintosh OS X. The same document viewed in Internet Explorer on Mac OS X is displayed in Figure 2. While the Internet Explorer view is more informative—displaying, as it does, the XML elements in a hierarchical view—this is rarely what a programmer would want the reader to see.

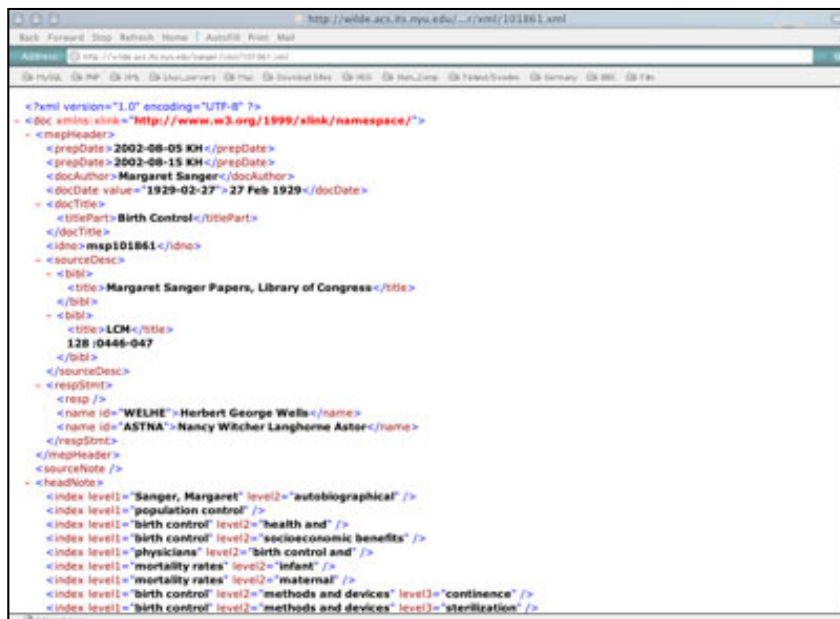


Figure 2. The same document viewed in Internet Explorer on Mac OS X.

Making XML More Reader-Friendly

W3C (The World Wide Web Consortium), the international Web standards body, has developed and is continuing to develop standards for the publication of XML files on the Web. One method is Cascading Style Sheets (CSS). If you are familiar with HTML, you most likely are aware of Cascading Style Sheets.

A Cascading Style Sheet is a file that allows you to define particular styles for HTML

elements. For instance, with CSS you can indicate that you want all `<h1>` elements to be a certain size, color, and style by specifying it in the style sheet. A typical style definition would look something like:

```
h1{text-align: center; font-size: 18px; font-weight: bold}
```

The same holds true for XML. You can use a CSS to indicate how certain tags should be displayed (or if they should be displayed at all). Figure 3 shows what our XML document looks like when it has been associated with a CSS. The result is more pleasing to the eye and in line with what a reader would expect.

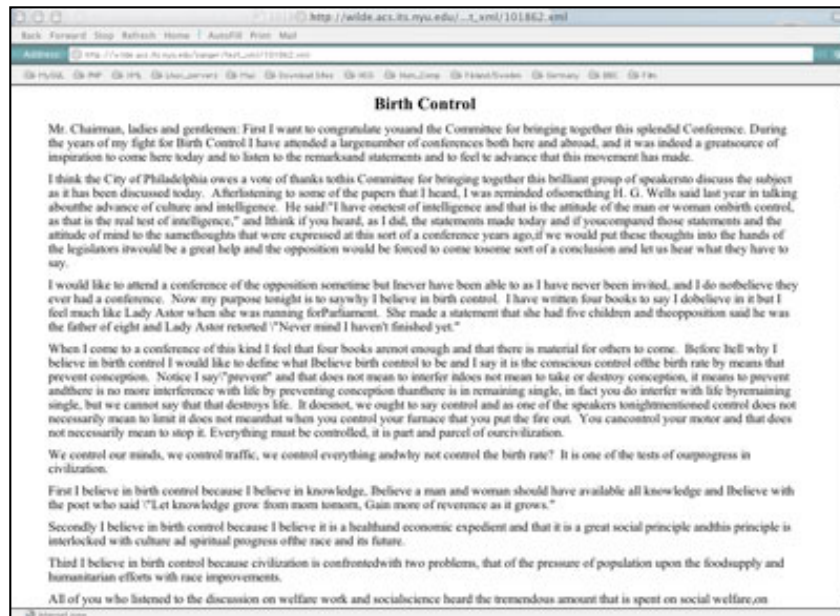


Figure 3. The sample document after a Cascading Style Sheet has been applied.

For many people, the use of CSS with their XML documents may suffice for their web publication needs. There are limitations, however. Your XML document can only display its information in the order in which it occurs in the document and, more important, you are relying on your reader to have a browser that supports CSS for XML. As of now, browser support of CSS for XML is a bit sketchy; therefore, it is risky to rely on client-side technology for your site to work.

There are other limitations to CSS. There are times when you may want to rearrange the order in which the information in your XML document appears. For instance, the Sanger documents contain a header with metadata (information about the data) concerning author name and date of publication.

```
<mepHeader>
<prepDate>2002-08-05 KH</prepDate>
<prepDate>2002-08-15 KH</prepDate>
<docAuthor>Margaret Sanger</docAuthor>
<docDate value="1929-02-27">27 Feb 1929</docDate>
<docTitle>
<titlePart>Birth Control</titlePart>
</docTitle>
<idno>msp101861</idno>
<sourceDesc>
<bibl>
<title>Margaret Sanger Papers, Library of Congress</title>
</bibl>
<bibl><title>LCM</title> 128 :0446-047</bibl>
</sourceDesc>
<respStmt>
```

```

<resp></resp>
<name id="WELHE">Herbert George Wells</name>
<name id="ASTNA">Nancy Witcher Langhorne Astor</name>
</respStmt>
</mepHeader>

```

This information occurs at the beginning of the XML document but needs to appear in different places when displayed. There are also situations in which you want particular conditions to control which data in the XML document is displayed. For instance, you could have an XML document containing a list of names and addresses. What if you want to display only those names and addresses from a certain city or zip code?

This is where XSLT comes in. XSLT stands for Extensible Stylesheet Language Transformations, a powerful style sheet language developed by W3C to work with XML. XSLT is too powerful and too complex to describe in great detail here, but it works like CSS, in that a style sheet is applied to an XML file.

It is also similar to CSS in that it operates on XML elements specified in the style sheet. However, it does much more than apply styles to these elements. It can apply actions to these elements, such as transforming the element name. It can use programming logic, such as "if/then" conditionals. Most important, it can transform an XML file and output it as another XML file or as an HTML, text, or PDF file, or even stream the output to another program, such as a web browser. Today, XSLT is typically used to transform an XML document into an HTML file that can be easily viewed in a web browser, eliminating any special client-side technology that would be needed to view the XML documents.

XSLT is based on templates. A template is created for a specified XML element. When the style sheet finds that element in the XML document, the template is applied. For instance, there is an element, `<unclear>`, used in the Sanger documents to indicate when a word is unclear in the original document. Perhaps the handwriting is illegible or the print has been smeared. In the Sanger XML file it looks like this:

```

<unclear>some text here</unclear>
<p>In the web browser it should display like this: [some text here?].
<p>The HTML code needed for this output is:
[<em>some text here</em>?]
<p>To complete this transformation, XSLT uses code that looks like this:
<xsl:template match="unclear">
[<em>
<xsl:apply-templates/>
</em>?]
</xsl:template>

```

This translates to: "whenever you come across the `<unclear>` element, print `` and then apply templates". This means search within the `<unclear>` element to see if there are any other elements that need templates applied. If not, just print out the text within the `<unclear>` element. Then print out `"?"`". This transforms `<unclear>some text here</unclear>` to `[some text here?]`, which, when viewed in a web browser, looks like: "[some text here?]

That was a very long explanation for such a simple thing, but once you learn XSLT and discover that you can do very powerful transformations, the promise of XML becomes clear. Your XML document truly becomes a data source from which you can produce any sort of document.

How do you perform these transformations? The style sheet does not just magically transform the XML file into another XML or HTML file. Theoretically, you should be able to link to any XSLT file from your XML document and allow the web browser to

perform the transformation in real time, similar to the way CSS works. This, however, is not the case in reality. Only certain web browsers—and, typically, only the most recent versions of these—successfully perform XSLT transformations. The hope is that someday all web browsers will have the built-in functionality to run XSLT transformations, but at present, you can't rely on this.

In the meantime, you need to use an intermediate piece of software called an XSLT processor that can read the XML file and the XSLT file and output a third file, in our case, HTML. When we were initially developing the style sheet for the Sanger project, we used an XSLT processor named Saxon. Saxon is written in Java and in its simplest form runs as a command-line program. If you are ambitious, you can write your own GUI, but for our purposes the command line worked well.

We used Saxon to test our style sheets by transforming some sample XML documents into HTML files and viewing the HTML files in a web browser. When we were satisfied with the style sheet, we began to look for a more robust solution, preferably one that would do the transformations on the fly. That is, we wanted a person to be able to visit the site and request a certain Sanger document, and have the XML file transformed on the fly and sent to the web browser as a stream of HTML.

That way, as more XML files were added to the collection or changes were made to the files, they would not have to be transformed manually into HTML, and the reader would always get the most recent versions of Sanger documents. Of course, if your documents will be static, you can do the XSLT transformations ahead of time (with Saxon or another processor) and simply store the HTML files on your server for viewing by the reader.

Publishing HTML from XML on the Fly

One possibility we considered was to write a Java applet that would be embedded in a web page and use Saxon to perform the transformations. After some research, however, we decided there was a solution that would require less application programming on our side. That solution was to use PHP with its XSLT extension.

What is PHP? To quote the PHP website (<http://www.php.net>), "PHP is a widely-used general-purpose scripting language that is especially suited to web development and can be embedded into HTML." It is typically used in conjunction with the MySQL database management system and Apache web server to create dynamic websites. It is similar to JSP and the commercial web applications Lasso, Active Server Pages, and Cold Fusion. It is also similar in some respects to Perl and Python.

We have been using PHP in the ITS Humanities Computing Group for over a year for other projects with great success, and it was natural for us to investigate its XSLT capabilities. Lucky for us, PHP can be configured to perform XSLT transformations.

The installation of PHP that comes with most Linux distributions does not have the XSLT extension enabled, so some configuration is needed. First, the Sablotron XSLT processor must be installed. The developers of the XSLT extension for PHP hope to support other processors in the future, but for now Sablotron must be used. Then, PHP must be recompiled and installed to take advantage of the XSLT processor. For detailed instructions on this installation process, see http://www.nyu.edu/its/humanities/docs/php_xslt.html. Although PHP is typically run on Linux and Unix servers, it can also be installed on Windows as needed.

Once PHP is configured for XSLT transformations, the coding is quite easy. The lines of code that perform the process are simply:

```

$xh = xslt_create();
xslt_process($xh,
"$sample.xml",
"sample.xsl", "sample.html");

```

The first line creates a new XML processor. This is used for returning errors and other feedback. The next line processes an XML document called "sample.xml", using an XSLT file called "sample.xsl", and outputs an HTML file called "sample.html". What makes PHP easy to use is that its code exists in a file with a .php extension but can be embedded in HTML and accessed through a web browser. This code can easily be added to other PHP and HTML code to produce a page that accepts variables and generates HTML on the fly.

The following is the full PHP code needed to publish an XML document on the fly, (with explanatory comments in *italics* for this example):

```

<?php
/* Allocate a new XSLT processor
$xh = xslt_create();
/* Process an XML document using the XSLT file named "test.xsl". The argument for
the XML document to be processed is not a static file name, but instead a variable
name. The variable value is passed via an URL to this PHP page (http://www.foo.edu/
transform.php?var=filename.xml). There is no HTML file output. Instead, the HTML is
output to the variable named "$result".
$result = xslt_process($xh, "$_GET[var]", "test.xsl");
/* If the transformation is successful ($result has a value) then print out $result. When
PHP prints $result, it is simply sending HTML to the browser as a stream.
if ($result) {
print $result;
}
/* If the transformation did not work, then print out this error message:
else {
print "Sorry, $_GET[var] could not be transformed by test.xsl into $_GET[var] the
reason is that .xslt_error($xh) and the error code is " .xslt_errno($xh);
}
?>

```

These few lines of code allow you to do on-the-fly, server-side XSLT transformations that create nicely formatted HTML documents suitable for the Web. With a little more code, our final product looks very nice indeed. Those awkward XML files we saw in the beginning are now transformed and displayed to the reader as shown in Figure 4.



Figure 4. The sample XML document after on-the-fly, server-side XSLT.

There are other solutions for publishing XML documents on the Web. You can work with the application program interfaces (APIs) provided with Saxon and develop your own server-side transformation system, but it involves more application development time and experience programming in C, C++, or Java. You can also use the Apache suite of products including Cocoon, the XML application server, but these products are still in their early stages of development and not 100% reliable, and also require working with an API to develop your own interface.

PHP is a nice solution because it is free, already installed on most Linux machines, also runs on Windows, relatively easy to configure for use with XSLT, very easy to code, and already works in conjunction with an Apache web server. With very little coding and configuration, you can create a fast and robust XML publishing system.

Resources

- XML: <http://www.w3.org/XML/>
- CSS: <http://www.w3.org/Style/CSS/>
- XSL & XSLT: <http://www.w3.org/Style/XSL/>
- PHP: <http://www.php.net>
- Saxon: <http://saxon.sourceforge.net/>
- Sablotron: <http://www.gingerall.com>

* Text source: *Margaret Sanger Papers, Library of Congress, LCM 128:0446-047.*

Author Biography

Matthew Zimmerman is a Humanities Computing Specialist in ITS' Academic Computing Services. He can be reached at matthew.zimmerman@nyu.edu.