Processing XML using PHP

Barry Cornelius
Computing Services, University of Oxford
Date: 8th November 2002; first created 4th November 2002
http://users.ox.ac.uk/~barry/papers/
mailto:barry.cornelius@oucs.ox.ac.uk

---

1 XML **1**
2 Some XML applications **3**
3 XSL Transformations **4**
4 Three approaches **4**
5 Using a standalone tool **5**
6 Executing PHP scripts from WWW pages **7**
7 Support in browsers **9**
8 Comparison of the three approaches **10**

---

1 XML

1.1 What is XML?

Although originally HTML was a language in which the details of how a WWW page was displayed by a browser was mainly left to the browser to decide, over the years HTML has evolved so that the author of a WWW page has more and more control over the appearance of the page.

This means that the HTML used for a WWW page is an intermingling mix of text giving the data to be displayed together with other text describing how it is to be displayed:

```
<table border=1>
   <tr>
      <td><b>inkjet cartridges</b></td>
      <td><em>HP Deskjet print cartridge 51645A</em></td>
      <td bgcolor="yellow">19.50</td>
   </tr>
...
</table>
```

XML is another markup language. One of the key differences is that an XML document does not usually contain presentation details: instead, it is just used to describe data. Another difference is that the tags of the language are chosen by the author rather than being determined by the language. For this reason, it is called the *eXtensible Markup Language*:

```
<consumables>
   <product>
      <category>inkjet cartridges</category>
      <item>HP Deskjet print cartridge 51645A</item>
      <price>19.50</price>
   </product>
   ...
</consumables>
```

The WWW page:
http://www.dur.ac.uk/barry.cornelius/papers/xml.processing.using.php/code/consumables.xml
contains a complete XML document. Elsewhere in this paper, a URL like this will be abbreviated to CODE/consumables.xml, where CODE is an abbreviation of:
http://www.dur.ac.uk/barry.cornelius/papers/xml.processing.using.php/code

1.2 Some jargon

Here is some jargon:

```
<price>19.50</price>
```

is called an *element*. It is introduced by a *start tag*. In this example, the start tag is:

```
<price>
```

And it is terminated by an *end tag*:

```
</price>
```

A start tag may have one or more *attributes* as in:

```
<price units="pounds" country="UK">19.50</price>
```

An element may be *empty* as in:

```
<price amount="19.50"></price>
```

and this can be abbreviated to:

```
<price amount="19.50"/>
```

Another example is:

```
<unknown_price></unknown_price>
```

and this can be abbreviated to:

```
<unknown_price/>
```

1.3 The main goals of XML

As I see it, there are three main goals of XML:

1. To provide a language that just describes some data. If it is necessary to describe how the data is to be displayed (and in some situations this is not the case), the description is given in a separate document.
2. To allow the author to use their own tags. In this way, it is possible for an author to choose tags that are appropriate to the data.
3. To insist on the language being used in a correct manner rather than being tolerant, e.g.,
   - each start tag must have an end tag;
   - the values of attributes must be quoted.

1.4 Well-formed, DTDs and valid

Although WWW browsers are lenient about the quality of the HTML that they receive, the XML specification does not allow XML parsers to be lenient. Any document that is not *well-formed* is not allowed to be fixed: it must be reported as being in error.

An XML document may use a *document type definition* (*DTD*). This can either appear in the same file as the XML document or it can appear in a separate file. The DTD is a sequence of rules describing the markup language that is used by the XML document.

```
<!ELEMENT consumables (product*)>
<!ELEMENT product (category, item, price)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

An example of a DTD is in the file `CODE/consumables.dtd`.

An XML document that conforms to a DTD is said to be a *valid* document.

Even if an XML document has a DTD, not all XML parsers actually check that the document conforms to the DTD.

1.5 XML schemas

More recently, *XML schemas* have been introduced. They can be used to define the structure of some XML (instead of using a DTD). Two of the main differences are:

1. An XML schema is written as an XML document (which means that there is less to learn and that it can be parsed by an XML parser).
2. When producing an XML schema, not only can you use predefined types, your schema can also introduce new types that help you to define the structure of your XML.

1.6 Resources about XML

1. Jon Bosak, ‘XML, Java and the future of the Web’,
   http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm
2. Elliotte Rusty Harold and W. Scott Means, ‘XML in a Nutshell (2nd edition)’,
   O’Reilly, 2002, 0-596-00292-0.
3. Mark Birbeck (editor) *et al*, ‘Professional XML (Programmer to Programmer) (2nd edition)’,
   Wrox Press, 2001, 1-861005-05-9.
4. Charles Goldfarb and Paul Prescod, ‘The XML Handbook’,
   Prentice Hall, 2000, 0-13-014714-1.
5. Some of the main WWW pages about XML are

   http://www.w3c.org/XML/,
   http://www.xml.com/, http://www.xml.org/, http://www.ibm.com/xml/,
   http://www.ibiblio.org/xml/, http://www.oasis-open.org/,
   http://www.oasis-open.org/cover/ and http://www.ucc.ie/xml/.

2 Some XML applications

As we have seen, XML allows you to produce a markup language to describe some data where you choose the tags to be used. Once you have chosen a set of tags, and perhaps have produced a DTD or a schema to describe the structure, you can then produce XML documents that use these tags. The act of producing a new markup language creates a new *XML application*.

Although we can all write our own XML applications, it becomes useful to establish agreed XML applications. Examples of XML applications that have been produced include:

1. CDF, Channel Definition Format,
   http://msdn.microsoft.com/workshop/delivery/cdf/reference/CDF.asp
2. CML, Chemical Markup Language,

   http://www.xml-cml.org/
3. ebXML, an XML for electronic business,

   http://www.ebxml.org/
4. GeoTech-XML, Geotechnical Engineering,

   http://www.ejge.com/GML/
5. MathML, Mathematical Markup Language,

   http://www.w3.org/Math/
6. SMIL, Synchronized Multimedia Integration Language,

   http://www.w3.org/AudioVideo/
7. SVG, Scaleable Vector Graphics,

   http://www.w3.org/Graphics/SVG/
8. TexML,

   http://www.alphaworks.ibm.com/tech/texml
9. XGL,

   http://www.xglspec.org/
10. XHTML, a rewrite of HTML as strict XML,

    http://www.w3.org/MarkUp/

There are many more. One list of XML applications is given at:
http://www.xml.org/xml/registry.jsp.

3 XSL Transformations

3.1 XSL

Often XML documents are provided on the WWW so that they can be read by other people. Suppose some other person wants to read our data, but suppose that their XML reading program requires only part of the data, or it requires the XML to be written in a different way. What would be useful is a way of describing how one XML document can be transformed into another one.

This forms one of the roles of the *eXtensible Style Language* (*XSL*). So an XSL document can be used to describe how to transform an XML document into some other XML document. This is called an *XSL Transformation* (*XSLT*). As HTML is reasonably close to XML, an XSL document is often used to describe how to transform an XML document into HTML.

3.2 The `consumables` element

The XML document shown earlier contained data describing some products and how much they cost. The outermost element of this XML document is the `consumables` element. Suppose we now wish to produce HTML from this XML document.

Here is part of an XSL document that can do this:

```
<xsl:template match="consumables">
   <html>
   <body>
   <table>
      <xsl:apply-templates/>
   </table>
   </body>
   </html>
</xsl:template>
```

This is a rule that says: when you come across a `consumables` element, output:

```
<html><body><table>
```

then process the elements between the `consumables`'s start tag and its end tag; and then output:

```
</table></body></html>
```

You can see that the notation that is used by an XSL document is XML, and so XSL is yet another XML application.

3.3 The `product` element

The other rule that we need in our XSL document is one that copes with a `product` element. For this we can use:

```
<xsl:template match="product">
   <tr>
   <td><xsl:value-of select="category"/></td>
   <td><xsl:value-of select="item"/></td>
   <td><xsl:value-of select="price"/></td>
   </tr>
</xsl:template>
```

This extracts the characters embedded in the `category`, `item` and `price` elements of a `product` element and places them in the cells of the HTML table.

3.4 Other aspects of XSL

A complete XSL document is given at `CODE/consumables.xsl`.

There are many other aspects to the use of XSL to transform documents. These include the possibilities of:

1. re-ordering the processing of the elements of an XML document;
2. looping over elements;
3. deciding which elements to include;
4. sorting with respect to some element.

4 Three approaches

Three approaches for processing an XML document will now be considered.

4.1 Approach One: using a tool to process XML

We could write a program to process an XML document. The program could be used to produce HTML which we could then make available on the WWW. More details about this are given in Section 5.

The problem with using a tool is that we have to remember to run it everytime the XML document is updated. The remaining two approaches enable the user of a browser to work directly on an XML document.

4.2 Approach Two: processing XML from a PHP script

If a webserver supports the running of server-side programs, then a browser visiting a page could trigger a webserver to run some code that reads an XML document and produces HTML. This could be done by using a CGI program or by running a servlet (coded in Java) (if your webserver supports the running of servlets). However, Section 6 of this document looks at how a PHP script can be used to read an XML document and produce HTML.

4.3 Approach Three: getting a browser to process XML

Increasingly, browsers are providing support for the use of XML and XSL. So Section 7 looks at how XML and XSL are supported by Internet Explorer 6 and Netscape 7.

5 Using a standalone tool

5.1 Creating a tool in the PHP language

From the early days of XML, it has been reasonably easy to write programs that read XML documents. To begin with, these programs were mainly written in Java, but now it is easy to write such programs in a number of programming languages (including PHP).

Although the code of a PHP file is normally triggered into action by a webserver, it is now possible to run standalone PHP programs, i.e., to run PHP programs from a Unix prompt or from a Windows command shell window. Although the support for this is experimental in PHP 4.2.3, it is fully supported in PHP 4.3.0 (which is currently only in pre-release). We now look at how to write a PHP program that processes an XML document.

The XML parser that comes with PHP allows you to write PHP code that walks through an XML document. It generates *events* for each part of a document, generating an event for each start tag, for each end tag, and for each occurence of characters appearing between a start tag and an end tag. In a PHP program, you can nominate functions to be called when these events occur.

Let's look at this in more detail. In a PHP program, you first need to create an XML parser:

```
$xmlparser = xml_parser_create();
```

You then register the functions that you wish to be called. For example, given:

```
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
```

the `startElement` function will be called when the parser meets a start tag; `endElement` will be called when it meets an end tag; and `characterData` will be called for the characters in between.

Having done all that, you then set the XML parser loose on your document:

```
$fp = fopen("consumables.xml", "r");
while ($data = fread($fp, 4096)) {
   xml_parse($xml_parser, $data, feof($fp));
}
```

Here is a complete program that reads `consumables.xml` and outputs an HTML table:

```
<%
   function startElement($parser, $name, $attrs) {
      if ($name == "consumables") {
         print "<table>";
      }
      else if ($name == "product") {
         print "<tr>";
      }
      else {
         print "<td>";
      }
   }

   function endElement($parser, $name) {
      if ($name == "consumables") {
         print "</table>";
      }
```

```
      else if ($name == "product") {
         print "</tr>";
      }
      else {
         print "</td>";
      }
   }

   function characterData($parser, $data) {
      print $data;
   }

   function startDocument() {
      print "<html>\n";
      print "<body>\n";
   }

   function endDocument() {
      print "</body>\n";
      print "</html>\n";
   }

   $file = "consumables.xml";
   $xml_parser = xml_parser_create();
   // insist that the tags have the right case
   xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, false);
   xml_set_element_handler($xml_parser, "startElement", "endElement");
   xml_set_character_data_handler($xml_parser, "characterData");
   if (!($fp = fopen($file, "r"))) {
      die("could not open XML input");
   }
   startDocument();
   while ($data = fread($fp, 4096)) {
      if (!xml_parse($xml_parser, $data, feof($fp))) {
         die(sprintf("XML error: %s at line %d",
                     xml_error_string(xml_get_error_code($xml_parser)),
                     xml_get_current_line_number($xml_parser)));
      }
   }
   xml_parser_free($xml_parser);
   endDocument();
%>
```

The `xml_parse` function walks through an XML document calling your three functions as it reaches appropriate points of the XML document.

Note that each part of the XML document is visited only once. If instead you want the ability to wander around an XML document, it is better to use the `xml_parse_into_struct` function: this will put a representation of the XML document into an array. PHP also has some DOM XML functions, but these are currently experimental.

5.2 Creating a tool to create HTML from XML and XSL

Instead of sprinkling the HTML we want to generate all over a PHP program, we could instead use the XSL stylesheet that was produced in Section 3.

PHP provides an extension that can do XSL transformations. In order to use this extension, the person installing PHP needs to build it with a library that implements XSLT. Currently, the only library supported by PHP is the Sablotron library, but future versions of PHP will support other libraries (such as Xalan and libxslt).

No matter what library for XSLT has been included, it will be used in the same way. Its use is demonstrated by the following program.

```
<%
   $xsltproc = xslt_create();
   $html = xslt_process($xsltproc,"consumables.xml","consumables.xsl");
   if (!$html) {
      die("XSLT processing error:" . xslt_error($xsltproc));
   }
   xslt_free($xsltproc);
   echo $html;
%>
```

The crucial statement of this program is:

```
$html = xslt_process($xsltproc,"consumables.xml","consumables.xsl");
```

The call of the `xslt_process` function takes the file `consumables.xml` and transforms it using the rules in the file `consumables.xsl`. This will produce a long string of HTML instructions, and this is stored in the `$html` variable. In the above program, the statement:

```
echo $html;
```

causes this HTML to be output.

5.3 Resources about XSL and XSLT

1. Dave Pawson, ‘XSL Frequently Asked Questions’,
   http://www.dpawson.co.uk/xsl/xslfaq.html

2. Ken Holman, ‘What is XSLT?’,
   http://www.xml.com/pub/a/2000/08/holman/index.html

3. Paul Sandoz, ‘FOP Slide Kit’,
   http://www.sun.com/software/xml/developers/fop/

4. Doug Tidwell, ‘Tutorial: Transforming XML documents [into HTML, SVG, and PDF]’,
   http://www.ibm.com/xml/

6 Executing PHP scripts from WWW pages

6.1 Producing HTML from `consumables.xml` and `consumables.xsl`

Instead of running a tool to transform an XML document into HTML, you could get a webserver to run a program that does the transformation. This can be done in many ways, including the use of a CGI script, a PHP script, a servlet using SAX, or by using a JavaServer page.

For example, we could have the following file of PHP code.

```
<%
   $xsltproc = xslt_create();
   $html = xslt_process($xsltproc,"consumables.xml","consumables.xsl");
   if (!$html) {
      die("XSLT processing error:" . xslt_error($xsltproc));
   }
   xslt_free($xsltproc);
   echo $html;
%>
```

If this file is put into a `public_html` directory, we are providing a PHP script that automatically generates HTML from an XML document.

6.2 Getting the XML from somewhere else

Accessing a file of XML assumes that some other process has already created it (and keeps it up to date). It may be more useful for this XML to be generated by some other means. If we want to do this, we have to call the `xslt_process` function in a different way. Just to demonstrate how this is done, here is a PHP script that has some XML stored in a string variable:

```
<%
   $xml =
        "<?xml version=\"1.0\"?>"
      . "<consumables>"
      . "   <product>"
      . "      <category>cassette tape</category>"
      . "      <item>DC6150 cartridge</item>"
      . "      <price>9.50</price>"
      . "   </product>"
      . "</consumables>";
   $arguments = array("/_xml" => $xml);
   $xsltproc = xslt_create();
   $html = xslt_process($xsltproc, "arg:/_xml", "consumables.xsl", NULL, $arguments);
   if (!$html) {
      die("XSLT processing error:" . xslt_error($xsltproc));
   }
   xslt_free($xsltproc);
   echo $html;
%>
```

6.3 Generating the XML from data stored in a database

It may be that the data we want is stored in a database. What we could do is as follows:

- extract the required data from the database;
- generate some XML from this data;
- transform this XML using an XSL document;
- display the resulting HTML.

For example, suppose we want to offer a WWW page where the visitor can look at a database containing some items displaying those items that have a cost less than some price. We will provide a WWW form for the visitor to type in the price.

Suppose we also want to offer the visitor a choice on how to display the data. We could provide radio buttons on the WWW form to get the visitor to choose between different styles of layout.

Here is a WWW form that achieves this:

```
<HTML>
   <BODY>
      <FORM METHOD="POST" ACTION="prices.php">
         What is your choice of price?<BR>
         <INPUT TYPE="text" NAME="somevalue">
         <BR>
         <INPUT TYPE="radio" NAME="layout" VALUE="plain"> plain
         <BR>
         <INPUT TYPE="radio" NAME="layout" VALUE="table"> table
         <BR>
         <INPUT TYPE="submit" VALUE="Submit price">
         <INPUT TYPE="reset">
      </FORM>
   </BODY>
</HTML>
```

When the visitor to this form clicks on the `Submit price` button, the `prices.php` script will be executed with two variables already set: the `$somevalue` variable will be initialised to the value that was typed in the form's textbox, and the `$layout` variable will be initialised to either `plain` or `table`.

We can use the value of the `$somevalue` variable in an SQL query (that produces a resultset):

```
mysql_connect("mysql.dur.ac.uk", "", "");
$SQLQuery = "SELECT * FROM consum WHERE price < $somevalue";
$result = mysql_db_query("Pdcl0bjc_prices", $SQLQuery);
$numrows = mysql_numrows($result);
for ($rownum = 0; $rownum<$numrows; $rownum++) {
   echo mysql_result($result,$rownum,"ID");
   echo mysql_result($result,$rownum,"goods");
   echo mysql_result($result,$rownum,"price");
   echo "<BR>\n";
}
```

Each iteration of the above for loop outputs three values. Instead of using the for loop to output these values, we could get it to produce XML-decorated forms of these values, e.g. to generate something like:

```
<product>
<ID>42</ID>
<goods>3.5" floppy disc</goods>
<price>0.35</price>
</product>
```

And we can get it to append these characters to the end of a string (being stored in a variable called `$xml`). This is illustrated by the `prices.php` script:

```
<%
   mysql_connect("mysql.dur.ac.uk", "", "");
   $SQLQuery = "SELECT * FROM consum WHERE price < $somevalue";
   $result = mysql_db_query("Pdcl0bjc_prices", $SQLQuery);
   $numrows = mysql_numrows($result);
   if ( $numrows == 0) {
      die("<p>There are no consumables with a price < $somevalue</p>");
   }
   $xml = "<?xml version=\"1.0\"?><consumables>";
   for ($rownum = 0; $rownum<$numrows; $rownum++) {
      $xml .= "<product>";
      $xml .= "<ID>".mysql_result($result,$rownum,"ID")."</ID>";
      $xml .= "<goods>".mysql_result($result,$rownum,"goods")."</goods>";
      $xml .= "<price>".mysql_result($result,$rownum,"price")."</price>";
      $xml .= "</product>";
   }
   $xml .= "</consumables>";
   $arguments = array("/_xml" => $xml);
   $xsltproc = xslt_create();
   $html = xslt_process($xsltproc, "arg:/_xml", $layout.".xsl", NULL, $arguments);
   if (!$html) {
      die("XSLT processing error:" . xslt_error($xsltproc));
   }
   xslt_free($xsltproc);
   echo $html;
%>
```

The above script arranges for the value of `$xml` to be passed to a call of `xslt_process`. In this call, the third argument is either the value `"plain.xsl"` or the value `"table.xsl"`.

The file `plain.xsl` could contain:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
   <xsl:template match="consumables">
      <html>
      <body>
      <pre>
         <xsl:apply-templates/>
      </pre>
      </body>
      </html>
   </xsl:template>
   <xsl:template match="product">
      <xsl:value-of select="ID"/>
      <xsl:value-of select="goods"/>
      <xsl:value-of select="price"/>
      <br/>
   </xsl:template>
</xsl:stylesheet>
```

and the file `table.xsl` could contain:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
   <xsl:template match="consumables">
      <html>
      <body>
      <table>
      <tr bgcolor="yellow">
      <td>ID</td>
      <td>goods</td>
      <td>price</td>
      </tr>
         <xsl:apply-templates/>
      </table>
      </body>
      </html>
   </xsl:template>
   <xsl:template match="product">
      <tr>
      <td><xsl:value-of select="ID"/></td>
      <td><xsl:value-of select="goods"/></td>
      <td><xsl:value-of select="price"/></td>
      </tr>
   </xsl:template>
</xsl:stylesheet>
```

6.4 Resources for using XML from PHP

1. Luis Argerich, Chris Lea, Ken Egervari, Matt Anton, Chris Hubbard, James Fuller and Charlie Killian, 'Professional PHP4 XML',
   Wrox Press, 2002, 1-861007-21-3.

7 Support in browsers

As time moves on, WWW browsers play catch-up with new developments. We now look at the support for XML and XSL that is provided in recent versions of browsers: we look at version 6 of Microsoft's Internet Explorer (IE6) and version 7 of Netscape's Navigator (NS7).

7.1 Use of CSS1 with XML

Both IE6 and NS7 provide support for XML with *CSS1* (version 1 of *Cascading Style Sheet*). To use CSS, you need to augment an XML document with a `xml-stylesheet` processing instruction. This results in the text that is in the file

`CODE/css1.xml`:

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<!DOCTYPE consumables SYSTEM "consumables.dtd">
<consumables>
...
</consumables>
```

The `type` attribute of this `xml-stylesheet` processing instruction says that this XML document is to be transformed using a CSS stylesheet and the `href` attribute gives the URL of the file containing the actual CSS instructions.

The file `css1.css` might contain:

```
consumables { display: block }
product     { display: block }
category    { font-size: x-large;
              color: black }
item        { font-size: large;
              color: red }
price       { background-color: yellow;
              color: blue }
```

7.2 Use of CSS2 with XML

CSS2 introduces some new aspects like the handling of tables. So instead we could use the file

`CODE/css2.xml`:

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="css2.css"?>
<!DOCTYPE consumables SYSTEM "consumables.dtd">
<consumables>
...
</consumables>
```

where the file `css2.css` contains:

```
consumables { display: table }
product     { display: table-row }
category    { display: table-cell; font-size: x-large;
              color: black }
item        { display: table-cell; font-size: large;
              color: red; text-indent: 0.1in }
price       { display: table-cell; background-color: yellow;
              color: blue; text-align: right; text-indent: 0.1in }
```

Although these instructions are obeyed correctly by NS7, they are not understood by IE6.

7.3 Use of XSL with XML

Both IE6 and NS7 can use XSL to process an XML document. You need to insert the following `xml-stylesheet` processing instruction in your XML document:

```
<?xml-stylesheet type="text/xsl" href="consumables.xsl"?>
```

The files

`CODE/xsl.xml` and `CODE/consumables.xsl`

contain the texts that can be used by both NS7 and IE6.

8 Comparison of the three approaches

Approach One uses a standalone tool to process an XML document. The resulting HTML document can then be put into an area that is visible from the WWW. The disadvantage of using Approach One is that you need to remember to run the tool each time the XML document is altered. However, this approach has the advantage that there are no prerequisites for either the webserver or the WWW browser.

Approach Two assumes that you can get your webserver to run a program to produce HTML from the XML document. Its advantage is that the HTML is always up-to-date as it is generated each time from the XML document. The first disadvantage of this approach is that you need a webserver that can run some appropriate program. Another disadvantage is that there may be some delay whilst the program is producing the HTML.

At the moment, Approach Three has one big problem: the browsers that are used by most visitors are not modern browsers and most of them are incapable of dealing with XML.