

---

# Serving XML: practical techniques for the dissemination of structured electronic information

---

*Ron Gilmour*

---

## The author

**Ron Gilmour** is Science Reference Librarian at the Science Library, University at Albany, Albany, New York, USA.  
E-mail: [gilmr@albany.edu](mailto:gilmr@albany.edu)

---

## Keywords

World Wide Web, Databases, Data structuring, Dissemination

---

## Abstract

The self-describing nature of data marked up using extensible markup language (XML) allows the XML document itself to act in a manner similar to a database, but without the large file sizes and proprietary software generally associated with database applications. XML data can be made directly available to users using a variety of methods. This paper explores methods for both server-side and client-side processing and display of XML-encoded data, using an annotated bibliography as an example.

---

## Electronic access

The research register for this journal is available at [http://www.mcup.com/research\\_registers](http://www.mcup.com/research_registers)

The current issue and full text archive of this journal is available at <http://www.emerald-library.com/ft>

## Introduction

Extensible markup language (XML, see Bray *et al.* (1998)) has given information providers the ability to structure their data in a format which is both human- and machine-understandable, and which is independent of any particular computing platform or processing application. But while XML has met with widespread approval within the mark-up world, there is no consensus on the best way to make XML structured data available to users. Much of the current use of XML relies on database software. While the combination of databases and XML can be very powerful, it also introduces redundancy. The self-describing nature of XML can make a database superfluous, especially for smaller applications. The current paper explores computationally lightweight, database-free means of communicating structured data via the World Wide Web.

## The database approach

Kim and Choi (2000) have described a bifurcated system composed of a “retrieval module” and a “browsing module”. Records are stored in an Access database. In the retrieval module, an active server pages (ASP) script generates XML documents that are then displayed to the user. The browsing module produces larger XML files that are displayed via extensible stylesheet language (XSL, see Clark (1999)). Chang (2000) created a searchable interface for archival finding aids by loading XML-encoded documents into an Access database and making that database searchable via ASP scripting. These approaches are complex and fail to take full advantage of an XML document’s ability to emulate a database through its own organization.

---

Many thanks to Marjorie Benedict of The University of Albany Libraries, who has allowed the author to use her bibliography on Gabrielle Roay as an example file for the techniques presented in this paper. The sample files may reflect outdated versions of his work. The most current version of the bibliography is available at <http://www.albany.edu/~mb648/Roy/>

## Non-database approaches

The flexibility of XML and the eagerness of various computing communities to adopt it have resulted in a number of ways to approach the problem of serving XML. These may be divided into server-side and client-side approaches.

The server-side techniques discussed here use the common gateway interface (CGI) protocol. Similar effects could be created using Java servlets or ASP. CGI allows server-side scripts to generate Web pages in response to a request from a client. When the script also incorporates an XML parser, it can selectively return information from an XML source document in response to parameters passed to it by the client. This approach has the advantage of familiarity – it is not substantively different from writing a CGI-based front-end to a database. The major disadvantage of this approach is slow response time, because it involves an ongoing dialog between the client and server, subject to all the hold-ups inherent in the Internet.

Client-side techniques have the advantage of placing the computational burden on the client rather than the server. In the client-side models described here, the complete XML source document is downloaded to the client and the desired information is extracted from it and presented to the user via either a Java applet or a browser-dependent script. See Figures 1 and 2 for a diagrammatic comparison of server-side and client-side methods.

### Server-side processing with CGI

Although the server-side approach exemplified by CGI scripting has some significant drawbacks that make it a poor choice if large amounts of traffic on a site are anticipated, it may be useful for small applications. This approach involves less intensive programming than do Java-based solutions and is very easy to implement.

The most common language for writing CGI scripts is Perl, which has a wide variety of tools available for use with XML data (Eisenzopf, 1998). Most of these are based on XML::Parser, a Perl module written by Larry Wall (the creator of Perl) using James Clark's Expat parser. These tools are freely available for

download from the Comprehensive Perl Archive Network (CPAN)[1].

In order simply to translate an XML document into HTML for display on the Web, one could write a script like `rstream.cgi` in Figure 3. This script creates an instance of the Expat parser and specifies a file for it to parse. The contents of the file are then accessed by a set of subroutines defined by the XML::Parser style "stream".

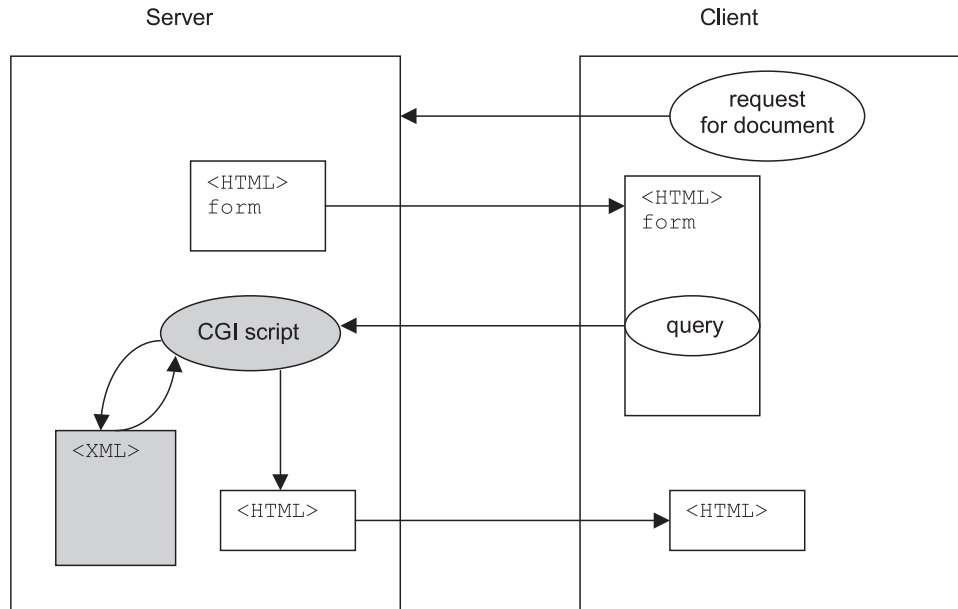
The "stream" style is simply a shorthand provided by XML::Parser to avoid having to list all of the individual subroutines in the parser invocation. "Stream" defines six subroutines that are invoked at different times during the parser's traversal of the document. The subroutines `StartDocument` and `EndDocument` are each called once at the beginning and end of the document being parsed. `StartTag` and `EndTag` are called whenever a start or end tag is encountered – these do much of the work in the program. The other two subroutines defined by "stream" are `Text` and `PI`, which provide methods for dealing with textual data and XML processing instructions, respectively. These methods are not used in `rstream.cgi`.

The example file being parsed is an annotated bibliography, consisting of entries containing author, title, source and annotation elements. The document's structure is defined by the simple document type definition (DTD) in Figure 4.

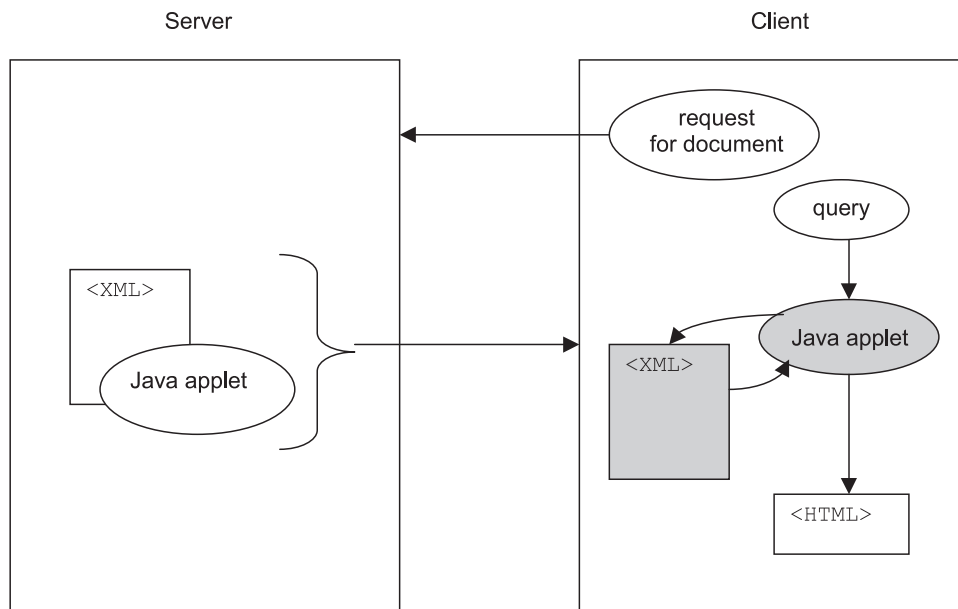
To make the annotations appear in a smaller font and with narrower margins in the resulting HTML, `rstream.cgi` defines the `StartTag` subroutine in Figure 5 such that on encountering an `<annotation>` tag, it produces a `<p class="annotation">` tag and, on encountering a closing `</annotation>` tag, a matching `</p>` tag is produced. The "annotation" style class is defined within a cascading style sheet (CSS) section created by the `StartDocument` subroutine.

This is a trivial example within the context of serving XML, because the results of such a simple transformation could more practically be saved as an HTML document and placed on a Web server normally, negating the need for CGI. This, in fact, is the most prudent decision if visitors to your site would only want to view a data set in a few predictable ways (e.g. a listing of staff that could be alphabetized by either first

**Figure 1** Server-side serving of XML: a query is submitted to the server via an HTML form; a CGI script parses the XML document, retrieves the requested information and returns the results to the client as an HTML document



**Figure 2** Client-side serving of XML: the entire XML document and an applet that can process it are sent in response to a single initial request. Individual queries from the user are processed by the applet on the client



**Figure 3** Parsing an XML document with XML::Parser

```
$parser = new XML::Parser(Style => "Stream"); # creates instance of XML::Parser  
$parser->parsefile("Royats.xml") or die "can't parse file\n"; # parses file
```

**Figure 4** A simple document type definition (DTD)

```
<!ELEMENT roy (entry+)>
<!ELEMENT entry (author, title, source, annotation?)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA|work)*>
<!ELEMENT source (#PCDATA|work)*>
<!ELEMENT annotation (#PCDATA|work)*>
<!ELEMENT work (#PCDATA)>
```

or last name). A truly dynamic means of serving the data is required only when there are more than a few alternatives, for example, if you want users to be able to query the data.

The files `rtree.cgi` and `rdom.cgi` provide examples of CGI-based search interfaces for XML data. Conceptually, `rtree.cgi` is the simpler of the two. Like `rstream.cgi`, it uses `XML::Parser`, but uses a style called “tree”. Rather than defining named subroutines, the tree style acts by reading the XML data into a multi-dimensional array from which data may be extracted later in the program. In other words, the tree structure of the XML document is transformed into another sort of tree structure (the multi-dimensional array) which Perl can use directly. `rtree.cgi` begins by identifying the author elements (see Figure 6). This bit of code may be translated as: “for each entry, get the third child (arrays in Perl begin numbering with ‘0’) of the third child of the *j*th child of the second child”.

The author to be searched is identified based on input from the user (via an HTML form) and is compared to the content of each author element (see Figure 7). Finally, when matching author nodes are found, additional information such as title is extracted (see Figure 8).

**Figure 5** StartTag subroutine

```
sub StartTag {
    my ($expat, $eltype) = @_; # get parameters passed to subroutine
    if ($eltype = "annotation") {
        print('<p class="annotation">');
    }
}
```

**Figure 6** Extracting author information in `rtree.cgi`

```
$limit = (@{$tree->[1]} + 1); # this is the number of entries
for($j=4; $j<$limit; $j+=2) {
    $author_node = $tree->[1][$j][2][2];
    ...
}
```

**Figure 7** Author search in `rtree.cgi`

```
$search_author = param( 'author'); # get search string from form
if(index($author_node, $search_author) >= 0) {
    # Does the string in $search_author appear as
    # part of the string contained in $author_node?
    print("$author_node");
    ...
}
```

**Figure 8** Extracting title information in `rtree.cgi`

```
$title = @{$tree->[1][$j][4]};
for($i=2; $i<=$title; $i+=2) {
    print("$tree->[1][$j][4][$i]");
}
```

An obvious drawback to this sort of code is the difficulty of reading and maintaining it, primarily due to the necessity of looping through multi-dimensional arrays. A solution is provided by the document object model (DOM) defined by the World Wide Web consortium (W3C, see Wood *et al.* (1998)). This is implemented in `rdom.cgi` through the `XML::DOM` module written by Enno Derksen. The code is considerably cleaner, with self-explanatory named functions replacing the

complex array references of `rtree.cgi` (see Figure 9).

Use of the DOM is somewhat controversial among XML application developers, since it carries with it a considerable computational burden. Like `XML::Parser`'s "tree" style, the DOM operates by creating a tree structure for the entire XML document in memory. This task probably accounts for much of the slowness of both `rtree.cgi` and `rdom.cgi`. Nevertheless, the DOM is an important standard which is already familiar to many Web developers and programmers from its use with dynamic HTML. Using the DOM also makes it easy to change parsing software without having to revise one's code extensively (McLaughlin, 2000).

#### Client-side processing I: Java

From the beginning, it has seemed that XML is made for use with client-side applications. This has been expressed by the slogan "XML gives Java something to do" (Bosak, 1997). The model described by Bosak (1997) and Bosak and Bray (1999), involves downloadable data that comes with custom software for manipulating the data. This is in contrast to the typical Web environment, in which most data are downloaded in a "dumbed-down" form so that it can be viewed by widespread browser applications whose only function is to provide an attractive, human-readable interface.

Currently, the Java applet is the tool of choice for making software available over the Web, although Microsoft's ActiveX technologies offer similar capabilities. Much XML-related

software has been written in Java. The `XSearch.java` file provides a demonstration of how an applet may be used to interact with XML data. Here, the search interface is also part of the applet rather than an HTML form, as was the case with the CGI examples. The parser used is `Ælfred`, originally developed by Microstar and now available from Open Text Corporation. `Ælfred` is a very small parser which was designed to be easily bundled into an applet (see St Laurent and Cerami (1999, Ch. 13)).

The parsing of the document is similar to what was described under `XML::Parser`'s "tree" style, except that rather than a multi-dimensional array, entries in the bibliography are saved in a Java vector (essentially an array consisting of references to individual entry objects). The search method is also similar to the CGI examples. In a search by author name, the program steps through all of the entry objects, extracting the author from each and comparing this to the search string (see Figure 10). This solution to making a large XML document searchable is a bit harder to implement than the CGI solutions, but has the advantage of speed. It may take a moment for the applet itself to load, but once this is done, repeated searches can be performed very quickly.

#### Client-side processing II: XML-aware browsers and client-side scripting

One alternative client-side approach to XML processing is to rely on the browser. It can either interpret the XML document based on

Figure 9 A non-DOM versus DOM example

*non-DOM:*

```
for($j=4; $j<$limit; $j+=2) {  
    my $author_node = $tree->[1][$j][2][2];  
    ...  
}
```

*Versus DOM:*

```
my $authors = $doc->getElementsByTagName('author');
```

Figure 10 A Java search by author name

```
for (int i=0, i<entries.size(); i++) {  
    current = (Entry) entries.elementAt(i);  
    tempauthor = current.getauthor().toUpperCase();  
    if (tempauthor.indexOf(target) >= 0) {  
        displayText (current.getauthor() +  
                    "\n" + current.gettitle() +  
                    "\n" + current.getsource() +  
                    "\n" + current.getannotation() + "\n\n");  
    }  
}
```

information provided by a stylesheet, or parse the XML so as to make the data available to a client-side scripting language such as JavaScript or VBScript. Currently, this would only be a practical solution within an environment in which uniformity of browser software is enforced (e.g. in a corporate intranet).

It is possible that in the near future most users will be equipped with XML-aware browsers, at which point direct use of XML with stylesheets or scripts may be an acceptable solution. The file `royjs.html`[2] demonstrates how a simple JavaScript embedded in an HTML page may be used to present XML data in an XML-aware browser. An alternative approach may be seen by viewing the XML source file (`Royats.xml`) directly[3]. In this case, an XSL stylesheet has been linked from within the XML document and is used by the browser to format the XML data as HTML.

## Conclusion

XML has been billed as “future-proof” (e.g. Wilner, 1998). This means that because the format is self-describing, XML will be understandable to humans long after specialty software for reading it is obsolete (assuming, of course, that a means of reading Unicode/ASCII text is available). This makes it a more stable format for long-term data storage than proprietary database formats. In the preceding discussion I have argued that XML may also serve as a medium for data presentation if it is combined with a few simple tools. See Appendix for more examples.

This approach should appeal to those in the scholarly research community. It is an easy means of making data available directly, rather than just reporting on the data. As information becomes an increasingly valuable commodity, and as researchers demand easier and more complete access to data, Web users will become dissatisfied with receiving HTML digests of research data. Providing data in the form of XML allows users to manipulate the data for themselves, whether with tools provided by the author or with those that they create themselves.

The question of scaling for non-database XML solutions has yet to be addressed. While

the tools presented here work fairly well for small data sets (the example file used here is under 300KB, with about 500 entries), they are unlikely to scale up as effectively as database solutions. It is with large-scale applications that the marriage of XML with existing database technologies will likely become extremely important. In these cases, “active” subsets of XML data could be loaded into databases for use with ASP scripts for online browsing, or the database could take the central role, producing not XML “documents” in the usual sense, but streams of XML data that could be fed to other applications, including, perhaps, an XML “warehousing” module for safe, platform-independent archiving of data.

## Notes

- 1 <http://www.cpan.org/>
- 2 `royjs.html` is at <http://www.albany.edu/~gilmr/Roy/royjs.html>
- 3 `Royats.xml` is at <http://www.albany.edu/~gilmr/Roy/Royats.xml>

## References

- Bosak, J. (1997), “XML, Java, and the future of the Web”, Available: <http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm>
- Bosak, J. and Bray, T. (1999), “XML and the second generation Web”, *Scientific American*, Vol. 280 No. 5, pp. 89-93, available: <http://www.sciam.com/1999/0599issue/0599bosak.html>
- Bray, T., Paoli, J. and Sperberg-McQueen, C.M. (1998), *Extensible Markup Language (XML) 1.0*, available: <http://www.w3.org/TR/1998/REC-xml-19980210.html>
- Chang, M. (2000), “An electronic finding aid using extensible markup language (XML) and encoded archival description (EAD)”, *Library Hi Tech*, Vol. 18 No. 1, pp. 15-27.
- Clark, J. (1999), *XSL Transformations (XSLT) Version 1.0*, available: <http://www.w3.org/TR/1999/REC-xslt-19991116>
- Eisenzopf, J. (1998), “Perl-XML module list”, available: <http://www.cpan.org/modules/by-module/XML/perl-xml-modules.html>
- Kim, H. and Choi, C. (2000), “XML: how it will be applied to digital library systems”, *The Electronic Library*, Vol. 18 No. 3, pp. 183-9.
- McLaughlin, B. (2000), *Java and XML*, O'Reilly, Cambridge, MA.
- St Laurent, S. and Cerami, E. (1999), *Building XML Applications*, McGraw-Hill, New York, NY.

Wilner, E. (1998), "Preparing Web data with SGML/XML", *Information Today*, Vol. 15 No. 5, p. 54; No. 6, pp. 52-4; No. 7, pp. 39-42.

Wood, L. et al. (1998), Document Object Model (DOM) Level 1 Specification, version 1.0, available <http://www.w3.org/TR/REC-DOM-Level-1/>

## Appendix. Example files

### CGI examples:

rstream.cgi	<a href="http://cgi.albany.edu:6060/~gilmr/rstream.cgi">http://cgi.albany.edu:6060/~gilmr/rstream.cgi</a>
source code	<a href="http://www.albany.edu/~gilmr/Roy/rstream.html">http://www.albany.edu/~gilmr/Roy/rstream.html</a>
rtree.cgi	<a href="http://cgi.albany.edu:6060/~gilmr/rtree.cgi">http://cgi.albany.edu:6060/~gilmr/rtree.cgi</a>
source code	<a href="http://www.albany.edu/~gilmr/Roy/rtree.html">http://www.albany.edu/~gilmr/Roy/rtree.html</a>
rdom.cgi	<a href="http://cgi.albany.edu:6060/~gilmr/rdom.cgi">http://cgi.albany.edu:6060/~gilmr/rdom.cgi</a>

source code	<a href="http://www.albany.edu/~gilmr/Roy/rdom.cgi">http://www.albany.edu/~gilmr/Roy/rdom.cgi</a>
-------------	---

### Java example

roysearch.html	<a href="http://www.albany.edu/~gilmr/Roy/roysearch.html">http://www.albany.edu/~gilmr/Roy/roysearch.html</a>
source code	<a href="http://www.albany.edu/~gilmr/Roy/XSearch.java">http://www.albany.edu/~gilmr/Roy/XSearch.java</a>

### JavaScript example (must be viewed in Microsoft Internet Explorer 5.x)

royjs.html	<a href="http://www.albany.edu/~gilmr/Roy/royjs.html">http://www.albany.edu/~gilmr/Roy/royjs.html</a>
------------	---

### XSL example (must be viewed in Microsoft Internet Explorer 5.x)

Royats.xml	<a href="http://www.albany.edu/~gilmr/Roy/Royats.xml">http://www.albany.edu/~gilmr/Roy/Royats.xml</a>
roy.xsl (the stylesheet)	<a href="http://www.albany.edu/~gilmr/Roy/roy.xsl">http://www.albany.edu/~gilmr/Roy/roy.xsl</a>