Open Source Software and XML

Open source software is about having greater control over your computing environment. XML is about distributing your data and information unambiguously. Through this hands-on workshop you will learn why these things are true and important to the future of libraries.

Open source software

Open source software (OSS) is both a philosophy and a process. As a philosophy it describes the intended use of software and methods for its distribution. Depending on your perspective, the concept of OSS is a relatively new idea being only five or ten years old. On the other hand, the GNU Software Project -- a project advocating the distribution of “free” software -- has been operational since the mid ‘80’s. Consequently, the ideas behind OSS have been around longer than you may think. When people think about OSS they often think about “free”, but the term “free” should be equated with freedom, and as such people who use “free” software should be:

1. free to run the software for any purpose
2. free to modify the software to suit their needs
3. free to redistribute of the software gratis or for a fee
4. free to distribute modified versions of the software

Put another way the term “free” should be equated with the Latin word “liberat” meaning to liberate, and not necessarily “gratis” meaning without return made or expected. In the words of Richard Stallman, we should “think of ‘free’ as in ‘free speech,’ not as in ‘free beer.’”

Activity - Writing and reading MARC

In this first activity you will write, read, and download sets of MARC records. Through the process you will learn about MARC as a data structure, become familiar with an open source “toolbox” for manipulating MARC data as well as means of acquiring MARC data from remote servers using other sets of open source software.

MARC (MAchine-Readable Cataloging) is a data structure design by Henriette Avram of the Library of Congress in the 1960’s. It originally provided a means for sharing bibliographic data and ultimately printing catalog cards. As a data structure it consists of three parts: 1) the leader, 2) the directory, and 3) the bibliographic information. The leader provides metadata about the record. The directory is a map to the record’s contents. The bibliographic information is the heart of the matter.

Below is a single MARC record with carriage returns added for readability. Can you tell me the title of the item it describes? If so, then how did you know? Guess?

```
01089njm  22002897a 4500001000800000005001700008007001500025008004
41000400350021000819060045001029550009001470100017001560280029001
73035002100202040001300223050002800236245003200264260003800296300
00580033451100510039250502350044350000180067865000300069670000170
0726953000900743991004700752 5711878 19940618100827.2 sdubsmennmp
luu 940617s1983   xx uun              eng      9(DLC)   94752991
a7 bcbc corignew d3 encip f19 gy-genmusic    ans24    a  9475299
1  00 a24.0146-1 bIdiot Records    a(OCoLC)30621098    aDLC cDLC
00 aIdiot Records 24.0146-1 00 aOrigami h[sound recording].    a[
S.l.] : bIdiot Records, c[1983?]    a1 sound disc : banalog, 33 1
```

```
/3 rpm, stereo. ; c12 in. 0  aWritten, composed and performed by
Fay Lovsky. 0  aRamon -- Window across the street -- Disney dust
-- Columbus Avenue -- Sugar me Sam -- Palmtree luxury -- Californ
ia daze -- Don't feed the animals -- Never seem able (to say good
bye) -- Fuss & fight -- Tiger & I -- One more time.    aBrief rec
ord.  0 aPopular music y1981-1990. 1  aLovsky, Fay.    aTA28    b
c-RecSound hIdiot Records 24.0146-1 wMUSIC
```

Writing MARC records is not difficult, if you have very specialized software. There must be about two dozen open source "toolboxes" and MARC record editors available, and in this case we will take advantage of one called MARC::Record. Using it I wrote a simple MARC record writer called marcwrite. Together we will use it to create sets of MARC records. Here's how:

1. open up your terminal
2. connect to the remote host
3. run **`marcwrite`**
4. enter an author's name
5. enter a title
6. enter one or more subject terms
7. append your record to a file with a **`.mrc`** extension
8. go to step #3 until you're tired

Reading MARC records is not difficult, if you have very specialized software. For example, you can use the Unix/Windows cat command or just about any plain text editor, but the output will be less readable than the MARC record above. Try this:

9. cat your file (**`cat [filename]`**)

Using MARC::Record I wrote a simple MARC record reader called marcread. You can use it to display your data in a more human-readable form. When you install MARC::Record two programs get installed along with way. The first is marcdump. It outputs a more complete view of your data. The second is marclint. It can be used to check the integrity of your records' data structure, not their content.

Explore your MARC records some more. Use marcread, marcdump, and marclint in the same way you used the cat command, above.

Libraries have been co-operatively sharing their bibliographic data for decades if not a century. Over time MARC records have been centrally located and easily accessible, if you have very specialized software -- specifically a Z39.50 client. A "toolbox" called YAZ (Yet Another Z39.59 client), written and supported by Index Data, is such a thing. After installing YAZ you can connect to the Library of Congress, search their catalog, and save what you find as MARC data. Here's how:

10. run **`yaz-client`**
11. define where your data will be saved (**`set_marcdump [filename]`**)
12. connect to the Library of Congress (**`open z3950.loc.gov:7090/voyager`**)
13. do searches (**`find [query]`**)
14. display results (**`show all`**)
15. go to Step #6 until you get tired
16. exit the program (**`quit`**)

Once you have installed YAZ and downloaded sets of MARC data, you can use a program called yaz-marcdump to display the fruits of your labor. Use it in the same manner you used cat, marcread, marcdump, and marclint in the exercises above.

As an additional activity, download and install MarcEdit, a full-fledged MARC record editor written and maintained by Terry Reese (Oregon State University). Use it to create new records or edit the ones from the previous exercises.

The MARC record data structure contains the “bread and butter” of library content, but the structure is also is very specific to the profession. Fortunately there are a number of open source applications and “toolboxes” available for reading, writing, and acquiring MARC data.

Indexes make search easier

Indexes make searching for information easier. Take a book, for example. It contains a table-of-contents outlining things to come. Then there is the content itself, the body of the book. Finally, there is the back-of-the-book index. If you want to find a specific fact do you look in the table of contents? No, you use the index. Articulate the most appropriate word of the fact you are seeking, find the word in the index, turn to the page number. Repeat as necessary.

A library catalog is a kind of index. Specifically, it is an index to the things a library owns, licenses, or otherwise wants to provide access. Search the catalog for word or phrase. Identify pointers to the content (call numbers or now-a-days URL’s). Use the pointers to acquire the information. MEDLINE and ERIC are indexes, indexes to journal literature. Search the index for words or phrases, and get back citations. Google is an index, an index to Internet resources. Search Google. Get back a URL. Acquire content.

Manually creating an index is laborious. Read a text. Identify “important” words and concepts. Note their location in the text. Repeat until done. On the other hand, the information retrieval (IR) community has been creating computer-generated indexes for the past twenty or thirty years. These indexers work the same way as human indexers but more thoroughly. Instead of including only the only the “important” words or concepts all words are included. Moreover, computer generated indexes and their accompanying search engines provide more sophisticated means for search and display including Boolean logic, word stemming, sorted result sets, and relevancy ranking.

Indexes, whether created manually or automatically, are fundamental to the process of finding information, and therefore they are fundamental to the functions of libraries.

Client/server computing

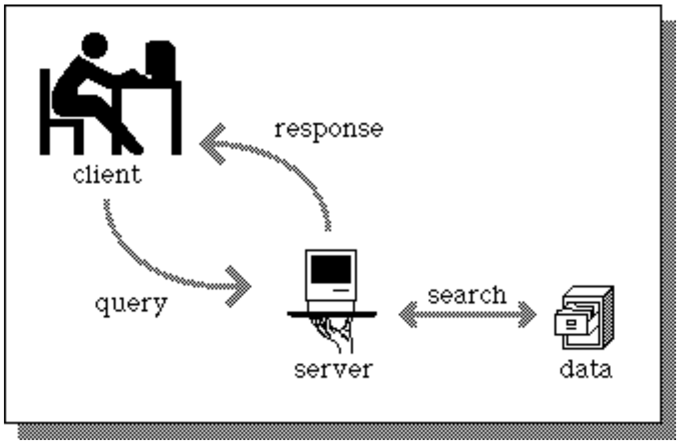
To truly understand how much of the Internet operates, including the Web, it is important to understand the concept of client/server computing. The client/server model is a form of distributed computing where one program (the client) communicates with another program (the server) for the purpose of exchanging information.

The client’s responsibility is usually to:

1. handle the user interface
2. translate the user’s request into the desired protocol
3. send the request to the server
4. wait for the server’s response
5. translate the response into “human-readable” results
6. present the results to the user

The server’s functions include:

1. listen for a client’s query
2. process that query
3. return the results back to the client



An illustration of client/server computing

Flexible user interface development is the most obvious advantage of client/server computing. It is possible to create an interface that is independent of the server hosting the data. Therefore, the user interface of a client/server application can be written on a Macintosh and the server can be written on a mainframe. This allows information to be stored in a central server and disseminated to different types of remote computers. Since the user interface is the responsibility of the client, the server has more computing resources to spend on analyzing queries and disseminating information. This is another major advantage of client/server computing; it tends to use the strengths of divergent computing platforms to create more powerful applications.

In short, client/server computing provides a mechanism for disparate computers to cooperate on a single computing task. A lot of Internet-based open source software exploits client/server computing because client/server computing makes it easier to implement things the “Unix Way”.

Activity - A simple OPAC

In this activity you will create simple online catalog (OPAC). In other words, you will index MARC records and providing access to the index. Through this process it will become apparent that creating collections and describing them through the use of MARC is only half of the problem to be solved. The second half is about providing access to the collections and then services against them.

Indexing MARC data is a very tricky process because it first needs to be parsed and then interpreted. Remember, it was really never intended to be the content for electronic indexes, instead it was intended for card printing. Most integrated library systems vendors have made their living by creating MARC record indexers, but an open source “toolbox” called Zebra, again by Index Data, is available for “free”. Ironically, many ILS vendors use the YAZ and Zebra toolkits in the products they sell to libraries. After installing Zebra it is easy to index your MARC data and provide access to it via Z39.50. Here’s how:

1. open up your terminal
2. connect to the remote host
3. create a set of MARC records (**`cat [filenames] >> catalog.marc`**)
4. edit a configuration file (see, zebra.cfg)
5. index your data (**`zebraidx -g catalog update data/catalog.marc`**)
6. run a Z39.50 server (**`zebrasrv localhost:9999`**)

You will now want to search your index, and you can use yaz-client:

7. run **`yaz-client`**
8. open up a connection to your server (**`open localhost:9999`**)

9. search (**`find [query]`**)
10. display (**`show all`**)
11. go to Step #9 until you get tired
12. exit (**`quit`**)

MARC::Record, YAZ, and Zebra are all “toolboxes” allowing you to create your own applications. Just as it was possible to use MARC::Record to create marcwrite and marcread, it is possible to use YAZ to create a Z39.50 client. With that in mind, here’s another irony. In computerize these “toolboxes” are known as modules or “libraries”.

Using YAZ, I wrote a simple Z39.50 client in Perl, and I called it zoom.pl. Give it a try:

13. open up your terminal
14. connect to the remote host
15. search (**`zoom.pl [query]`**)

Despite the utter simplicity of the terminal-based yaz-client and zoom.pl programs, people are not going to want to search your OPAC with them. Instead they (and you) will want to use a Web browser. No problem, the next step is to create a CGI (Common Gateway Interface) script -- a client/server application -- against your Z39.50 index. Based on zoom.pl, I wrote zoom.cgi. To use it open up your Web browser, go to http://boole.uvt.nl/zoom/zoom.cgi, and enter queries:

Maintaining systematic descriptions of collections, and then providing services against them through an index is the technical foundation of traditional librarianship. Using open source software you can efficiently provide these services in our increasingly global networked environment.

Databases for data storage and maintenance

As alluded to previously, libraries love to create lists. In a digital environment, the most effective way to maintain lists are through relational database applications. Access and Filemaker are popular desktop database applications. Oracle is probably the best known “big gun” database application. MySQL and Postgres are increasingly popular open source software solutions.

The concept of a “relational” database was first articulated in the late 1970’s and early 1980’s from folks at IBM. It’s core concept -- “normalization” -- breaks downs sets of information into their most discrete parts and saves it in subsections called “tables”. Then, by creating pointers from one table to another it is possible to establish “relationships” between data elements and “join” them together to create reports. Because of normalization information in relational databases is saved in one place and one place only. Consequently, there is no need for global find/replace operations. Change the value once and all subsequent reports will contain the up-to-date information. Relational databases also remove the need for field like subject_01, subject_02, subject_03, etc. In relational databases any record can have any number of repeatable fields, and they don’t all need to be articulated before-hand.

MyLibrary is an open source software “toolbox” applied against a relational database schema making it easy to create and maintain much of traditional library-oriented content. The core of a MyLibrary instance are a set of four tables. The first table is called resources, and it’s fields are a superset of the Dublin Core elements (title, creator, description, etc.). The second table is librarians and contains the names and contact information of information specialists. The third table is for patrons and it too includes name/contact information. The last set of tables are for creating simple faceted classification schemes -- a controlled vocabulary. Using this database design it is possible to create sets of information resources, librarians who maintain these resources, people who use these resources, and relationships between all three through a common vocabulary. MyLibrary is written in Perl and makes it easy to read and write to the underlying database; it is one of many “frameworks” for building digital libraries.

OAI-PMH - a de-centralized OCLC

Again, libraries love to create lists. These lists usually consist of metadata describing items in their collections. The protocol called OAI-PMH (Open Archives Initiative-Protocol for Metadata Harvesting) is efficient way to share these lists in a de-centralized manner. As a computer protocol, OAI-PMH describes an agreed upon communication process, and believe it or not, there are only six things, called "verbs", the protocol describes:

1. **Identify** - This verb is used to verify that a particular service is an OAI repository.

2. **ListMetadataFormats** - Meta data takes on many formats, and this command queries the repository for a list of meta data formats the repository supports.

3. **ListSets** - This verb is used to communicate a list of topics or collections in a repository.

4. **ListIdentifiers** - It is assumed each item in a repository is associated with some sort of unique key -- an identifier. This verb requests a lists of the identifiers from a repository.

5. **GetRecord** - This verb provides the means of retrieving information about specific meta data records given a specific identifier.

6. **ListRecords** - This command is a more generalized version of GetRecord. It allows a service provider to retrieve data from a repository without knowing specific identifiers.

OAI-PMH is a client/server process. The client (called the "service provider" in OAI-PMH parlance) sends the server (the "data repository") one of the commands and their qualifiers. The server does some computing and returns a stream of metadata in XML to the client. It is then up to the client to reformulate the metadata and provide some useful service against it.

OAI-PMH was initially designed to assist the open access publishing process, but it has taken on a life of its own and can be used to share just about any kind of metadata, if not the actual data itself.

Activity - Being an OAI service provider

In this activity you will combine a relational database, MyLibrary, and OAI-PMH to create a searchable/browsable "digital library". Specifically, we will exploit OAI-PMH to download and automatically classify electronic journal titles from the Directory of Open Access Journals (DOAJ). To do this you will use a script called doaj2mylibrary.pl. Here's how:

1. open up your terminal
2. connect to the remote host
3. navigate to the MyLibrary terminal interface (**`cd terminal-interface`**)
4. collect journal metadata (**`./doaj2mylibrary.pl`**)

Once run, doaj2mylibrary.pl will create a MyLibrary location called URLs, create a set of facet/term combinations depending on the OAI-PMH sets of the DOAJ, download all of the DOAJ's metadata, and save it to the underlying MyLibrary database. To see the fruits of your labors you can use a terminal-based interface to MyLibrary (main-menu.pl) or a Web-based interface:

5. explore a terminal-based interface to MyLibrary (**`./main-menu.pl`**)
6. browse a Web-based admin interface (http://boole.uvt.nl/mylibrary/admin/)
7. browse a Web-based user interface (http://boole.uvt.nl/mylibrary/user/)

Using the same process you can download sets of metadata describing images using a program called images2mylibrary.pl:

8. in the terminal, collection image metadata (**`./images2mylibrary.pl`**)

As above, you can see the fruits of your labor through the terminal and Web-based interfaces.

Browsing is nice, but to make finding information you need a... index. In this case we will use an older open source indexer called swish-e, maintained by Bill Mosely. A script index-resources.pl called does the work, but it is not fast. Try this:

9. index your data (**`./index-resources.pl`**)

After a few minutes you should be able to search the index through a script called search.pl. There is no Web-based search interface in this activity.

10. search your index (**`./search.pl`**)

Activity - Being an OAI data repository

In this activity you will serve content via OAI-PMH. Specifically, you will convert MARC records into simple OAIXML files (with a program called marc2oai), make them available with a "toolbox" called XMLFile created by Hussein Suleman, and then use the same process with content from MyLibrary. Here's how:

1. open up your terminal
2. connect to the remote host
3. navigate to the marc directory (**`cd marc`**)
4. run marc2oai (**`./marc2oai [filename] [prefix]`**)

You should now be able to use your Web browser to peruse the data at http://boole.uvt.nl/oai/. There you will find a CGI script, a configuration file, a set of Perl modules, and a set of data where all of the OAIXML files you just created reside. You should also be able to browse the repository using the OAI Repository Explorer.

Remember, one of the biggest strength of database is the ability to write reports against them. In this case we will write a sets of reports against MyLibrary to create OAIXML files for every record in a MyLibrary instance. We will use a program called mylibrary2oai.pl. Here's how to use it:

5. navigate to the MyLibrary terminal interface (**`cd terminal-interface`**)
6. run mylibrary2oai.pl (**`./mylibrary2oai.pl`**)

Like above, you should now be able to use your Web browser to see the files it created as well as use the OAI Repository Explorer as an OAI service provider.

Relational databases make it easy to store and manipulate data. Indexes make information easily findable. Community standards reduce the need to re-invent the wheel. Open source software builds on these concepts and allows others to improve upon them for the benefit of all.

XML in libraries

In a sentence, the eXtensible Markup Language (XML) is an open standard providing the means to share data and information between computers and computer programs as unambiguously as possible. Once transmitted, it is up to the receiving computer program to interpret the data for some useful purpose thus turning the data into information. Sometimes the data will be rendered as HTML. Other times it might be used to update and/or query a database. Originally intended as a means for Web publishing, the advantages of XML have proven useful for things never intended to be rendered as Web pages.

XML documents have syntactic and semantic structures. The syntax (think spelling and punctuation) is made up of a minimum of rules:

1. XML documents always have one and only one root element - The structure of an XML document is a tree structure where there is one trunk and optionally many branches. The single trunk represents the root element of the XML document.
2. Element names are case-sensitive - Each of the following possible elements names are different: name, Name, NAME, nAmE.
3. Elements are always closed - Each element is denoted by opening and closing brackets, the less than sign (<) and greater than sign (>), respectively.
4. Elements must be correctly nested - Consecutive XML elements may not be opened and then closed without closing the elements that were opened last first.
5. Elements' attributes must always be quoted - XML element are often qualified using attributes. For example, an integer might be marked up as a length and the length element might be qualified to denote feet as the unit of measure. For example: <length unit='meter'>5</length>. The attribute is named unit, and it's value is always quoted. It does not matter whether or not it is quoted with an apostrophe (') or a double quote (").
6. There are only five entities defined by default (<, >, &, ", and ') - Certain characters in XML documents have special significance, specifically, the less than (<), greater than (>), and ampersand (&) characters. The first two characters are used to delimit the existence of element names. The ampersand is used to delimit the display of special characters commonly known as entities; they ampersand character is the "escape" character. Because ' and " are used in attributes they have pre-defined entities as well: ' and ", respectively.
7. When necessary, namespaces must be employed to eliminate vocabulary clashes - The concept of a "namespace" is used to avoid clashes in XML vocabularies.

The semantics of an XML document (think grammar) is an articulation of what XML elements can exist in a file, their relationship(s) to each other, and their meaning. Ironically, this is the really hard part about XML and has manifested itself as a multitude of XML "languages" such as: RSS, RDF, TEILite, DocBook, XMLMARC, EAD, XSL, etc.

XML information is made accessible to humans as well as computers through an XML-based technology called XSLT (eXtensible Stylesheet Language: Transformation). XSLT is an XML "language". Write/create an XML file. Write an XSLT file. Use a computer program to combine the two to make a third file -- the transformation. The third file can be any plain text file including another XML file, a narrative text, or even a set of sophisticated commands such as structured query language (SQL) queries intended to be applied against a relational database application. Again, XSLT is a programming language. It is complete with input parameters, conditional processing, and function calls. Unlike most programming languages, XSLT is declarative and not procedural. This means parts of the computer program are executed as particular characteristics of the data are met and less in a linear top to bottom fashion. This also means it is not possible to change the value of variables once they have been defined.

Activity - Beyond MARC

MARC was an innovative data structure for its time, but considering today's computing environment it has all but outlived its usefulness. These exercises demonstrate a way the profession can begin the process of migrating way from MARC to something XML-based.

We begin by converting MARC data into MARCXML data using a program called marc2xml that was installed with the MARC::Record "toolbox". Here how:

1. open up your terminal
2. connect to the remote host
3. navigate to the marc directory (**`cd marc`**)
4. display MARCXML (**`marc2xml [marcfile] | less`**)
5. save MARCXML (**`marc2xml [marcfile] > [marcxmlfile]`**)

Through the use of XSLT technology (specifically, an open source program called xsltproc) you can convert your MARCXML data into MODS using a stylesheet from the Library of Congress:

6. display MODS (**`xsltproc MARC21slim2MODS3.xsl [marcxmlfile] | less`**)
7. save MODS (**`xsltproc MARC21slim2MODS3.xsl [marcxmlfile] > [modsfile]`**)

To make the data searchable you need to... index it. Using a third open source "toolbox" (called Kinosearch by Marvin Humphrey) we can index our data using a program called mods2kinosearch.pl. Try this:

8. index your data (**`./mods2kinosearch.pl [modsfile]`**)

Finally, you can search your index in two ways: 1) you can use a terminal-based interface or 2) your Web browser:

9. search from the terminal (**`./kinosearch.pl [query] | less`**)
10. search from your browser (http://boole.uvt.nl/kinosearch/)

In 1965, MARC was ahead of its time, but it's time has past. XML is so much more flexible and expressive. Moreover, XML is used by a much larger number of people compared to the library community. MARC as a data structure is limiting, and it should be considered a legacy format.

Activity - Writing XML

The "X" in XML stands for "eXstensible". This essencially means you are able create your own mark-up language as long as it is consistent with the seven syntax rules. In this exercise you will create a mark-up language for letters (correspondance) and use XSLT to transform the letter into XHTML. Go:

1. as a group, discuss the necessary (XML) elements of a letter
2. use your Web browser to copy letter.xml, letter.dtd, letter2html.xsl, and letter.css to your desktop from http://boole.uvt.nl/letterml/
3. use a text editor to edit your local copy of letter.xml making sure it conforms to the DTD (letter.dtd)
4. view your edited letter in your browser
5. go to Step #3 until your letter is well-formed

By adding XML commands (called "directives") to XML files you can make them load external files, such as cascading stylesheets or XSL stylesheets. Through this process you can make the XML more human-readable.

6. add the following text to your letter so it is the first line:

```
<?xml-stylesheet href='letter2html.xsl' type='text/xsl'?>
```

7. view your letter in your browser
8. view the source code of your letter in your browser and notice how the XML has been transformed into HTML
9. replace the first XML directive with the following:

   ```
   <?xml-stylesheet href='letter.css' type='text/css'?>
   ```

10. view your letter in your browser again and notice how the XML has not changed but the CSS rendered the XML in a human-readable form

As an added exercise, copy your letter to the server and validate it against the DTD using xmllint, or better yet, install xmllint on your desktop computer and validate it there.

Creating your own XML can be quite exciting and empowering, but it usually not a good idea because someone has probably already created an XML schema that does what you desire.

Flavors of XML

There is little reason to design your own mark-up language because somebody has probably already invented on to meet your needs. Some of the more common languages for libraries and other cultural heritage institutions include:

- **EAD (Encoded Archival Description)** - EAD is an SGML/XML vocabulary used to markup archival finding aids. As you may or may not know, finding aids are formal descriptions of things usually found in institutional archives. These things are not limited to manuscripts, notes, letters, and published and published works of individuals or groups but increasingly include computer programs and data, film, video, sound recordings, and realia. Because of the volume of individual materials in these archives, items in the archives are usually not described individually but as collections. Furthermore, items are usually not organized by subject but more likely by date since the chronological order things were created embodies the development of the collections' ideas. Because of these characteristics, items in archives are usually not described using MARC and increasingly described using EAD.

- **MARCXML** - MARCXML is vocabulary supporting a "roundtrip" conversion of MARC data to XML. MARCXML has a place for each and every MARC bibliographic field, indicator, subfield, and well as the MARC leader. It is entirely possible to convert a MARC record (or file of MARC records) into a MARCXML file (or file of MARCXML records). Similarly, it is entirely possible to go from a MARCXML file to a MARC file. In neither case will any data be lost in translation. Thus, it is supports "roundtrip" conversions.

- **MODS** (Metadata Object Description Schema) - MODS is a bibliographic schema with library applications in mind. Its elements are a subset of MARC 21 (and therefore MARCXML). It is designed to be more human-friendly than MARC 21 relying on language-based element names as opposed to numeric codes. Instead of denoting a title the code 245 in MARC/MARCXML it is called "title" in MODS. In short, it is a schema designed to build on the good work of MARC and traditional cataloging, and at the same time it is designed to meet the needs of today's environment.

- **RDF (Resource Description Framework)** - RDF is for encoding metadata in an XML syntax. RDF is very much like the idea of encoding meta data in the meta tags of HTML documents. Using the HTML model, meta tags first define a name for the meta tag, say, title. Next, the content attribute of the HTML meta tag is the value for the title, such as Gone With The Wind. For example, the following meta tag may appear in an HTML document: <meta name='title' content='Gone With The Wind'/>. These name/value pairs are intended to describe the HTML document where they are encoded. HTML documents have URL's. These three things, the name, the value, and the URL form what's called, in RDF parlance, a triplet. RDF is all about creating these triplets; it is all about creating name/value pairs and using them to describe the content at URLs.

- **TEI (Text Encoding Initiative)** - TEI is a grand daddy of markup languages. It is used most often used by the humanities community to mark up literary works such as poems and prose. Many times these communities digitally scan original documents, convert the documents into text using optical character recognition techniques, correct the errors, and mark up the resulting text in TEI. Ideally a scholar would have on hand an original copy of a book or manuscript along side a digital version in TEI. Using these two things in combination the scholar would be able to very thoroughly analyze the text and create new knowledge.
- **XHTML** - XHTML a pure XML implementation of HTML. Therefore the seven rules of XML syntax apply: there is one root element, element names are case-sensitive (lower-case), elements must be closed, elements must be correctly nested, attributes must be quoted, and the special characters must be encoded as entities. There are a few XHTML vocabularies ranging from a very strict version allowing no stylistic tags or tables to a much more lenient version where such things are simply not encouraged. XHTML documents require a DOCTYPE declaration at the beginning of the file in order to specify what version of XHTML follows.

Activity - Full-text indexes

People's expectations regarding the access to information have increased with the inception of the Internet. Now, more than ever, people expect to the the content of a thing, not just a pointer to it. These exercises demonstrate was full-text XML documents, specifically TEI files, and be transformed, full-text indexed, and made accessible via the Web. To get us started, a small set of TEI files have placed on a Web server:

1. browse TEI files (http://boole.uvt.nl/etexts/tei/)

You can validate these files for well-formedness as well as against their DTD with a program called xmllint:

2. open up your terminal
3. connect to the remote host
4. navigate to the etexts directory (`cd apache/htdocs/etexts`)
5. list the contents of the tei directory (`ls tei`)
6. validate a TEI file (`xmllint --valid --noout [teifile]`)

As an extra exercise, edit one of the TEI files, intentionally make it invalid, and repeat Step #6. As an extra, extra exercise download and install xmllint to your desktop computer and validate documents there.

Raw XML files are not necessarily intended for human consumption. XHTML and PDF files are better suited for this purpose. Using xsltproc and a stylesheet called tei2html.xsl we can transform our TEI files into files intended for Web browsers:

7. display XHTML (`xsltproc etc/tei2html.xsl [teifile] | less`)
8. save XHTML (`xsltproc etc/tei2html.xsl [teifile] > [htmlfile]`)

Typing the entire xsltproc command is long and tedious. A tiny script (tei2html.sh) has been written that does the transformation more easily as well as saves the resulting XHTML files in one nice, neat location:

9. save XHTML, again (**`bin/tei2html.sh [teifilenameroot]`**)
10. browse the XHTML (http://boole.uvt.nl/etexts/html/)

The process of creating the PDF files is similar. First the TEI files need to be transformed into an XML-based page-layout format called FO (Formatting Objects). Second, a FO processor (in this case a "toolbox" called fop -- Formatting Objects Processor) is used to convert the FO file into a PDF file. Here we use a stylesheet called tei2fo.xsl. Here's the hard way:

11. display FO (**`xsltproc etc/tei2fo.xsl [teifile]`**)
12. save FO (**`xsltproc etc/tei2fo.xsl [teifile] > fo/[fofile]`**)
13. convert FO to PDF (**`fop.sh fo/[fofile] pdf/[pdffile]`**)
14. browse the PDF (http://boole.uvt.nl/etexts/pdf/)

Here's the easy way using a shell script called tei2pdf.sh:

15. transform TEI to FO to PDF (**`bin/tei2pdf.sh [teifilenameroot]`**)
16. browse the PDF (http://boole.uvt.nl/etexts/pdf/)

You have now created a rudimentary browsable interface to a collection of full-text documents. To create a searchable collection you need to... I'm not going to say it. Using the Kinosearch "toolbox" again, a simple indexer called tei2kinosearch.pl was created. Try this:

17. index (**`bin/tei2kinosearch.pl`**)

You should now be able to query the full-text of the TEI documents using terminal-based script called kinosearch.pl:

18. search (**`bin/kinosearch.pl [query] | less`**)

Again, nobody is going to want to search your index using a terminal-based interface. That is so 1980's. Using SRU (Search/Retrieve via URL), a Web-based protocol akin to the venerable Z39.50, you can not only search your index and link directly to texts, but you can share access to your index seamlessly across the Web. To make this happen an SRU server (called sru-server.cgi) was created. To access is, simply:

19. use the SRU client (http://boole.uvt.nl/etexts/sru/client.html)

As an added exercise use the "view source" function of your Web browser to look at the response from the SRU server. It is XML and note how your Web browser functions as an XSLT processor to transform a linked stylesheet from the XML into XHTML for display.

The more full-text content you have available the easier it is to provide useful searchable/browsable interfaces to it. Having your content saved as XML makes the process even easier.

Web Services

Web Services are a combination of client/server computing, the Internet, and XML. The concept is simple. Over the Internet, one computer sends another computer a URL or a stream of XML. The second computer uses the URL or XML as input, does some processing, and returns to the first computer a stream of XML. The first computer finally takes the XML and transforms it for human or computer consumption. OAI-PMH is an example of a Web Service. So are SRU and the use of RSS created by blogs. This foundation provides for many opportunities:

1. Since the shapes of the HTTP requests and XML streams are expected to be similar from service

to service it is easy to create services that do similar things, such as provide the definitions of words, search indexes, display weather information, etc. Because these service should be implemented similarly, it should be easy to swap out one index search for another index search, for example. It is easy to create standards-compliant services.
2. Since the input of Web Services are HTTP requests and XML streams, Web Services computing does not favor any particular computer language or operating system.
3. Since the output of Web Services includes just information and no presentation layer, the output can be transformed for a wide variety of uses. The most easily understood is HTML, but the output could just as easily be transformed into PDF, email, RSS, an integer, a word, provide the input to update a relational database, or a channel for a portal application.
4. Since the goals of libraries are to collect, organize, archive, and disseminate data, information, and knowledge, it makes a lot of sense for libraries to exploit the Web Service technique in order to accomplish their goals, especially in a globally networked computer environment.

Activity - Creating a "mash-up"

These activities demonstrate how you can query remote Web Services, get XML back, and use the results to suppliment other information services. The first activity explores a dictionary Web Service:

1. append a word to the URL root (http://services.aonaware.com/DictService/DictService.asmx/Define?word=):
2. use your browser to send the query
3. notice the XML nature of the response
4. go to Step #1 until you get tired
5. open up your terminal
6. connect to the remote host
7. navigate to the juice/bin directory (**`cd apache/htdocs/juice/bin`**)
8. run dictionary.pl with the same word(s) from Step #1 (**`./dictionary.pl [word]`**)
9. notice how the XML response has been transformed into a simple XHTML ordered list

This activity demonstrates a rudimentary spell-checker:

10. append a (incorrectly spelled) word to the URL root (http://spell.ockham.org/?word=), send it with your browser, notice the XML nature of the response, and repeat until you get tired
11. explore these three different "clients" written against the spell server. In each case a form is presented, a query is sent, an XML response is returned, and the result is displayed:
    a. rudumentary client (http://boole.uvt.nl/spell/dumb/)
    b. hyperlink to many spellings (http://boole.uvt.nl/spell/smart/)
    c. Did You Mean? against the British Library SRU server (http://boole.uvt.nl/spell/british-library/)
12. explore Tomato Juice, a mash-up of a number of Web Services to create a more well-rounded information service (http://boole.uvt.nl/juice/)

XML is a very well-established data structure used by a huge and growing number of information providers. It enables many different types of data to be manifested as information. By distributing XML over the Internet (specifically via Web servers), it is possible to mix and match content to better meet people's information needs.

Workshop summary

The combined use of open source software and XML are the current means for getting the most out of your computing infrastructure. Their underlying philosophies are akin to the principles of librarianship. They enable. They empower. They are flexible. They are "free". The way to get from here to there is through a bit of re-training and re-engineering of the way libraries do their work, *not what* they do *but how* they do it. Let's not confuse the tools of our profession with the purpose of the profession. If you think libraries and librarianship are about books, MARC, and specific controlled vocabularies, then your future is limited. On the other hand, if you think libraries are about the collection, organization, preservation, and dissemination of data, information, and knowledge, then the future is quite bright.

External links

This is a simple list of external links from the workshop. It is presented here because the handout is intended to be printed, and the URL's get lost in the process:

- **MARC::Record** (http://search.cpan.org/dist/MARC-Record/) - A Perl “toolbox” for reading and writing MARC data
- **YAZ** (http://indexdata.com/yaz/) - A “toolbox” and set of binary files for implementing Z39.50 clients.
- **MarcEdit** (http://oregonstate.edu/~reeset/marcedit/html/) - A full-fledged MARC record editor
- **Zebra** (http://indexdata.com/zebra/) - An indexer/search engine combination
- **MySQL** (http://www.mysql.com/) - A relational database application
- **Postgres** (http://www.postgresql.org/) - Another relational database application
- **MyLibrary** (http://dewey.library.nd.edu/mylibrary/) - A digital library framework and “toolbox”
- **OAI-PMH** (http://www.openarchives.org/) - The cononical site for the Open Archives Initiative
- **Directory of Open Access Journals** (http://www.doaj.org/) - A list of open access journals
- **swish-e** (http://www.swish-e.org/) - Another indexer/search engine
- **XMLFile** (http://www.dlib.vt.edu/projects/OAI/software/xmlfile/xmlfile.html) - A “toolbox” for sharing metadata via OAI
- **OAI Repository Explorer** (http://re.cs.uct.ac.za/) - A sophisticated OAI service provider
- **xsltproc** (http://xmlsoft.org/XSLT/xsltproc2.html) - An XSLT processor
- **Kinosearch** (http://www.rectangular.com/kinosearch/) - A third indexer/search engine
- **xmllint** (http://xmlsoft.org/xmllint.html) - An XML validator
- **FO** (http://www.w3.org/TR/2001/REC-xsl-20011015/slice6.html) - The home page for Formatting Objects
- **fop** (http://xmlgraphics.apache.org/fop/) - A FO processor
- **SRU** (http://www.loc.gov/standards/sru/) - The home page of the Search/Retrieve via URL standard

---

Author: Eric Lease Morgan <emorgan@nd.edu>