

Welcome Back!

The ACCC Post

Everyone

CONTENTS

1 Welcome Back!
The ADN Post

2 XML and the
Future of the
Web

4 XHTML:
Between HTML
and XML

4 Web Resources
on XML and its
protocols

5 Processing
XML: XSLT

8 SOAP: XML
Under the
Covers

9 RSS: Spreading
the News

11 The Head
Crash

FaCT Faculty Computers

Academic Affairs has committed substantial funds toward putting an up-to-date computer on the desk of every tenured and tenure-track faculty member on campus *and* to replacing them on a regular basis in the future, as they become obsolete.

This program, the [Faculty Computer Trade-in](#) program or **FaCT**, was announced on June 26, 2000, by UIC Provost and Vice Chancellor for Academic Affairs Elizabeth Hoffman.

The ACCC will administer the FaCT program, providing every tenure-track faculty member with a new personal computer—your choice of a well-equipped Windows personal computer or a slightly less well-equipped Mac (Macs are a bit more expensive)—on a rotating 3- or 4-year cycle. The FaCT computers will come with a network connection, hardware service, and access to the ACCC Server Services shared-software program.

FaCT will be phased in over a 3-year period, with the third of UIC's faculty most in need of an update receiving new computers the first year. The FaCT Web site has all the details, including a description of the FaCT machines and links to a list of eligible faculty and to an automated facility that helps us determine who the "neediest" faculty are:
<http://www.uic.edu/depts/accc/hardware/fact/>

AHEAD Help Desk

[AHEAD](#), [ACCC Help and Answer Database](#), is the ACCC's new Web- and email-based problem database. It's on the Web at:

<http://consult.accc.uic.edu>

or click the [AHEAD \(HelpDesk\)](#) link in the "Quick Start" section on the left on the ACCC home page:

<http://www.accc.uic.edu>

We're using AHEAD for all ACCC help email

addresses now, including consult@uic.edu. Get ahead with AHEAD! (Sorry about that!)

The neatest thing about AHEAD is that it allows you to log and reply to problems on the Web, as well as by email, so you may choose whichever method you prefer. Or both, when that's convenient. AHEAD's Web interface allows you to see all of the communication about all of your recent problems, at any time, from anywhere, even if you don't have a working email system. (That's good if your problem is that your Eudora is broken!)

To make sure that only you can access your problems, when you use AHEAD on the Web, you'll have to login using your ACCC netid and password. (Just like Nessie and most other ACCC, UIC, and U of I personalized Web services.)

Even if you post your problem on the Web, you'll receive email from AHEAD when we reply to it. email from AHEAD looks a bit different from the Problem Database email you may have gotten from us in the past. The subject is [ProblemDB], then a 4-digit number and a brief description of the problem. The number is your [AHEAD problem ticket number](#). And there's a URL at the bottom of each AHEAD message, which will take you directly to all the correspondence about that problem on the Web.

If you decide to reply by email, be sure to reply to the note you receive from AHEAD. Sending a new note opens a new problem, which won't help. And please don't include more of the note you're replying to than you have to. Some problem tickets already have gotten a bit too long to handle.

So, to log a new problem or to view all of your recent problems, go to: <http://consult.accc.uic.edu> Or just send email to consult@uic.edu as always.

Continued on page 10

System Icons:



The Internet and the World Wide Web



MS Windows

Apple Macintosh
Readership Icons:

Everyone



Novice



Expert

Warning: High Risk of Acronym Overload!

But don't panic yet; most of the acronyms/initialisms in this article are discussed in the following articles. Or check this newsletter out online; all the blue words are links to further info there.

<http://www.accc.uic.edu/newsletter/adn28/>

XML and the Future of the Web

News on the Net

WWW Everyone

That's right, a new acronym: **XML**, e**X**tensible **M**arkup **L**anguage. Sure, buzzwords come and buzzwords go, but XML is going to be with us for quite a while. It's more than a replacement for HTML; it will enable structured information exchange and interactivity. Are you at all interested in how the Web is going to evolve? XML has reached critical mass.

Ancient History: Plain Text

Our purpose in this article is to answer the question: What is XML? But let's start another question: What is text and how do we use it? Does anyone remember Gopher? A Gopher page was plain text, pretty much like a page from a book, except it used only one font. No bold, no italic, no larger or smaller type, no wrapping to fit your window, no hyperlinks, no images. Everything was displayed exactly as if it had been typed on a typewriter. You recognized the title because it was at the top of the page. Maybe there was something that looked like a name near the title; from its position and the fact it looked like a name, you could guess that was the page's author.

The point is that only the human brain could extract information from a Gopher text page. A computer couldn't do it; it would have no standard way to break the page into its constituent parts. Not only is this lack of structure bad for computer applications and databases, it's not all that good for humans. You couldn't change the display to fit your window size, or change the default font, or find that one specific stock price to display on your Palm Pilot.

Although modern Web pages look a lot better than Gopher pages did, Web browsers still can't adapt their display *based on content*. This means no real interactivity, no e-commerce, no business-to-business transactions, *at least not without different custom applications for each case*.

Recent History and the Web

HTML was a good start toward organizing the information on a page. HTML introduced the concept of **markup** as a way to designate which parts of the text were which. HTML has `<h1>` for a big heading, `<h2>` for a smaller heading, `<p>` for

paragraph, `` for boldface, `` to change fonts, `
` for a line break, `<tr>` for a row in a table, and so on. And two crucial tags: `<a>` for links and `` for images. These tags give the Web its hypertext navigation and its graphical look. Finally, a browser had a fighting chance at adaptive display.

As good as HTML is, it has two significant drawbacks. The first is that HTML is not easily **extensible**. Mathematicians can't add a `<polynomial>` tag, chemists can't add `<benzene>`, and stockbrokers can't add `<stockprice company="cisco">376</stockprice>`. New tags had to be pre-understood by the browser, so authors were stymied.

Of course, given the competition between Netscape and Microsoft, new tags *were* added to each browser's HTML in incompatible ways. These new tags only made life worse, because now authors could write HTML that will work well on one browser but poorly on the other. And it is still impossible to add enough tags to suit everyone without making both the tag set *and* the browsers much too large and cumbersome.

The second drawback of HTML is the nature of the tags themselves, coupled with the desire of advertisers-cum-Web-authors to control the look and feel of their Web pages in every detail. That is, the tags have become largely layout-oriented, rather than meaning-oriented. If a designer decided that `<h1>` was usually rendered in too large a font, he'd use some combination of ``, ``, and `<center>` instead. While this means that the browser will display the page more to its designer's liking, it also means there is no `<h1>` tag that can be interpreted as an "important heading" for search and retrieval. Designers also are too often tempted to control the width and placement of text, not allowing the end user to make use of wider windows or higher-resolution screens.

SGML and the Solution

What to do? The answer was clear to those in the field, even before the popularity of HTML was apparent. That is, there needs to be a separation between *the tags in a document* and *the rules for*

dealing with those tags. Roughly speaking, a browser needs to download a set of rules for each Web page to use in rendering that particular page. This would allow authors to invent new tags because they could also specify the rules for dealing with those tags. It would also encourage designers to put styles, fonts, and layout into the rules, not the tags. This way, a given document could be rendered for different displays by changing the rules, not by changing the markup tags.

Fortunately for the Web's creators, **SGML**, **Structured Generalized Markup Language**, predated HTML; in fact, SGML inspired HTML. The answers to HTML's problems are also coming from SGML.

However similar the names, HTML and SGML are quite different. HTML is a set of markup tags (and more-or-less a set of rules for interpreting the tags). SGML, on the other hand, is a meta-language for defining general tag sets. In fact, the HTML tag set is now defined in SGML. There are any number of other **instances** of SGML tag sets. I wrote the one we use for the ACCC home page, for example, and I might write another one tomorrow.

SGML provides a standard way of saying which tags belong in a set, what **attributes** the tags have, and what arrangement of tags constitute a valid **document**. It lets you specify **syntax** (how to produce a valid document), but it does not deal with **semantics** (what the tags mean).

Enter XML

The Web world could have just accepted SGML, but it didn't, because SGML is complicated in ways that many thought were unnecessary. Instead, the **W3C**, **World Wide Web Consortium**, the Web's standardizing body, convened a committee to simplify SGML. That committee gave birth to XML in 1998.

XML, like SGML, is a language for specifying tag sets. It lacks some of the SGML bells and whistles that were off-putting to the Web community. XML is being embraced not only by Netscape and Microsoft, but also by a host of other companies, for all sorts of applications that may never involve a Web browser.

One encouraging aspect of current XML activity is the development of related standards. XML is not the solution by itself; it's only a very good start. XML provides a way to mark parts of a document with arbitrary tags, so that a generic XML parser can identify these parts. It does not, by itself, say what these parts mean or what to do with them.

For that, we need other standards: **style sheets** (rules that tell browsers how to **render** the text), such as **XSL** and **CSS**, and rules that specify links between documents, such as **XLink**. XML extensions are also needed, such as XML **namespaces**, which allow the merging of different XML tag sets in the same document, or XML **schemas**, which allow authors to specify exactly what constitutes a valid document.

And then there's **DOM** and **SAX**, which are ways to model documents. There are specialized tag sets such as **MathML**, **MatML**, and **ChemML**, not to mention the crucial **XHTML**—a bridge that is both HTML and XML that you can use now. Of course, there are new protocols that use XML, such as **RSS** and **SOAP** and **ebXML**.

Netscape and XML

Mozilla is the open-source version of Netscape's Web browser. The upcoming, next-generation version of Mozilla uses XML in two interesting ways. It supports the rendering of arbitrary XML documents, styled with CSS style sheets. This lets you go way beyond publishing customary HTML. It also lets you customize the browser itself, with an XML configuration file written in **XUL**, **eXtensible User interface Language**. That's right, you can use XML to change the browser's "chrome", the parts of the browser normally outside of the page you are downloading. One could conceivably build new non-browser applications, with new chrome, based on the Mozilla Gecko rendering engine, without rewriting Gecko's internals.

And Getting Back to Plain Text

As complicated as some of these developments may become, one very simple and attractive characteristic of XML won't change—*an XML document is text*. Marked text, to be sure, but readable with *any* word processor or editor. And this is a very good thing. Got old WordStar files? You've got a problem. Got old XML? You'll always be able to read it, even when you're using the latest Windows 2000 and the program that produced the XML is long gone. You won't need any special program, either, just whatever text editor is handy.

I've only scratched the surface in this article. And if all the acronyms I've given you are confusing, you're not the only one; many of them are still in development. I do know that XML and its cousins will change your cyberlife, I think for the better.

About these Articles

The articles about XML and its related protocols were written by Bob Goldstein, who, in one of his many guises, is head of the ACCC Web staff. Bob's an old hand at SGML (see the section "SGML and the Solution") and he's *really* enthusiastic about XML.

Any questions or comments? You can reach Bob at bobg@uic.edu.

XHTML: Straddling the Fence Between HTML and XML

News on the Net

XML is here, but full support in even the newest Web browsers is problematic, not to mention the older browsers that so many people still use that don't know anything about XML at all. And virtually all documents on the Web, including yours, are written in HTML, not XML.

Still, you want to get started on XML, without the pain of maintaining multiple versions of files. Maybe you have a compelling need to define your own tags or to use a specialized tag set for math or chemistry; if you do, you know who you are. More likely, you just want to make new documents (and convert old ones if it's not too hard) that will work with *both* traditional HTML-based browsers and new XML-based applications as they come along.

What's a poor developer to do? Yet another acronym: **XHTML**, [Extensible HyperText Markup Language](#).

- XHTML is an XML tag set, so XML tools work with it.
- XHTML can be written to work in existing HTML-based browsers. (Cool!)

WWW Everyone

- XHTML looks pretty much like regular HTML, so there isn't much to learn. You just have to be a bit more strict about your markup.

The defining document for XHTML1 (version 1 of XHTML) is on the W3C Web site:

<http://www.w3.org/TR/xhtml1/>

Fortunately, it's pretty readable. Some highlights:

- Tags must nest properly. For example:
BAD: `<p> ... </p>`
GOOD: `<p> ... </p>`
- Tag names must be lower case. XML is case sensitive, whereas HTML is not. So `<h1>` or `<H1>` is fine in HTML, but only `<h1>` is proper in XHTML.
- You must enclose attribute values in quotation marks. `` is ok in HTML, but for XHTML—actually for all XML—you have to do it like this: ``
- Most HTML elements have content, the “...” between the begin and end tags, e.g.,
`<p> ... </p>`

In XHTML, you must include both the begin tags and the end tags for tags that have content. So, if you use tags such as `<p>` or `` or `<td>`, you must explicitly use the corresponding `</p>`, `` or `</td>` appropriately. And that's just fine with browsers; in fact, it's good HTML programming.

- The HTML tags that don't have content, such as `
`, `<hr>`, and ``, are a problem. XML requires “empty tags” to be written as `
`, `<hr/>`, or ``

Unfortunately, most HTML browsers will choke on `
`. The good news is you can write it as `
` with the extra blank space before the “/” part. That's valid XML *and* it will satisfy existing browsers.

There are a few other considerations, but you can see that coding to XHTML standards is pretty easy. Be sure to check the W3C Web site for other links such as validators. It's not too early to get started with XML. Nor is it particularly difficult.

Web resources on XML and related protocols.

<http://www.xml.com>

The first place to look, the O'Reilly XML reference and resource page.

http://zvon.vscht.cz/ZvonHTML/Zvon/zvonTutorials_en.html

Zvon's XML/XSL tutorial page; lots of examples; another good place to start.

<http://www.w3c.org>

Web standards of all sorts; XML, CSS, XHTML, and more.

<http://www.xml.com/pub/Guide/XSL>

The O'Reilly XML.com page on XSL and related projects.

<http://www.xmlhack.com/>

A news site for XML developers.

<http://www.xml.com/pub/2000/04/05/feature/index.html>

The O'Reilly XML.com page on processing XML with perl.

<http://www.webreference.com/authoring/languages/xml/rss/>

WebReference.com's page on RSS; tutorials, tools, RSS collection sites.

<http://www.soap-wrc.com/webservices/default.asp>

The SOAP—WebServices Resource Center.

<http://msdn.microsoft.com/workshop/xml/articles/xmlmanifesto.asp>

XML Manifesto by the developer of SOAP; explains why XML is good for data transfer.

<http://www.ebxml.com/>

Main page for ebXML effort — B2B (business to business) standards.

<http://www.mozilla.org/xpfe/xp toolkit/xulintro.html>

Mozilla.org's XUL site; it has lots of details on XUL.

Processing XML: XSLT, a Matter of Style

News on the Net

If you want the benefits of XML, and there are many, you have to pay the price. That means finding a way to do something useful with your XML files. The available options are a bit more limited now than XML is itself, but they are improving rapidly. Fortunately, one of the big benefits is the ability to write your XML content now and change your mind repeatedly about how to process it, without changing the original data at all.

One option is to transform an XML document into some other format, be it non-XML, a different XML tag set, reorganized XML content, or HTML. If you were a programmer, you'd be looking into [DOM](#), [Document Object Model](#), or [SAX](#), [Simple API for XML](#), which break an XML document into parts in different ways, suitable for a program in Java, perl, or C++.

But in this article, I'll give you a tiny taste of [XSLT](#), [X Stylesheet Language-Transformations](#), which is the currently working part of the overall [XSL](#), [Extensible Stylesheet Language](#), standard. XSLT is a special XML tag set, but one that contains rules for processing other XML documents, rather than real content itself.

Style Sheets

Remember style sheets? The whole original premise of markup was the separation of documents into content (in this case, the XML) and styles, the rules for dealing with content. In simple terms, you'd send an XML file and a style sheet to a browser and have the document rendered. To change the rendering, just change the style sheet. And it actually can work that way. But when you look at the details of XML styles, things are much more flexible and useful than that simple picture.

HTML Style Sheets—CSS

The current champ of style-sheet languages is [CSS](#), [Cascading Style Sheets](#). CSS predates XML and is used to specify how HTML is to be rendered. For example, you could write a CSS rule that says “when

WWW Everyone

you see a `<p>` tag, set the text in 12 pt Helvetica” or “when you see ``, set the text in red but don't boldface.” Sure, you could do most of what CSS can do with judicious use of ``, `<table>`, and other HTML tags. But if you want to change the rendering of dozens of HTML files in one fell swoop, CSS is the way to go. CSS also provides many fine-control positioning commands and, best of all, has some support in most current browsers.

Unfortunately for CSS, rendering isn't the only thing you might want to do with a style sheet. You might want to insert new content or rearrange the old content. This CSS can't handle; it's just for rendering.

Rendering in the XML world:

CSS will have a competitor in [XSL-FO](#), [XSL-Formatting Objects](#), an XML style-sheet language for rendering. But the XSL-FO standard is not finished and there is no browser support at all.

XML Style Sheets—XSLT

Suppose you have an XML file with information about your department, such as the names and addresses of the faculty and staff, a list of grad students for each prof, a list of courses and their descriptions, and so on. Now you want to generate a list of professors in HTML to use on your departmental home page. Sure, you can copy and re-edit (and re-alphabetize) the file, but do you really want to do that every time you add a new professor? Or when you decide to reformat the output to include a list of courses by each prof? Better to write a style sheet, and use the style sheet rules to do the extraction and rearrangement.

Enter XSLT.

An Example Is Worth 1000 Tags

So here's an example: the XML file in figure 1.

The file is short, but the idea is clear—a `<department>` contains a set of departmental info such as `<dept-name>` and `<address>` and also a list of `<person>`s in random order. Each `<person>` can have a different type, which is indicated with the attribute `type`: `<person type="prof">`. And then there are subelements, such as `<name>` or `<house>`. An easy way to store information, provided you can process it.

Now suppose you just want a list of the people in the department. Names only, alphabetized. You want it in HTML format, with the name of the department at the top, suitable for use with your departmental home page. And please indicate somehow the departmental chair in the list.

Now consider figure 2, a simple XSLT file. It may look vaguely like a programming language, and in some ways it is. But it is fundamentally an XML document like any other and is not too hard to read, at least for simple cases.

An XSLT file has a bunch of `<xsl:template>` elements. Each such template is just a rule: "As you parse the XML file of interest, when you come across a tag of such-and-such type, do the following." So each template has to specify which tags it applies to (using the `match` attribute) and then specifies what actions to take in the body of the template element.

`<xsl:template match="dept-name">`

This is the simplest tag. This says, "When you come across an element of type `<dept-name>`, output `<h1>`, then the value of the `<dept-name>` element, then `</h1>`." That's it; simply put the name of the department into an `<h1>` element.

Figure 1: The XML Input File

```
<?xml version="1.0"?>
<department>
  <dept-name>
    Department of Magical Implications
  </dept-name>
  <address>
    10 Hogwarts Castle
  </address>
  <person type="chair">
    <name>Albus Dumbledore</name>
  </person>
  <person type="prof">
    <name>Severus Snape</name>
    <interests>potions</interests>
  </person>
  <person type="prof">
    <name>Minerva McGonagall</name>
    <house>Gryffindor</house>
  </person>
</department>
```

Figure 2: The XSLT Input File

```
<xsl:transform
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0"
  xmlns:saxon="http://icl.com/saxon"
  exclude-result-prefixes="saxon"
>
<xsl:template match="department">
  <HTML>
  <BODY>
  <xsl:apply-templates select="dept-name" />
  <UL>
  <xsl:apply-templates select="person" >
    <xsl:sort select="name" order="ascending" />
  </xsl:apply-templates>
  </UL>
  </BODY>
  </HTML>
</xsl:template>
<xsl:template match="dept-name">
  <h1>
    <xsl:value-of select="."/>
  </h1>
</xsl:template>
<xsl:template match="person">
  <li>
    <xsl:value-of select="name"/>
  </li>
</xsl:template>
<xsl:template match="person[@type='chair']">
  <li>
    <xsl:value-of select="name"/>
    (<b>Chair</b>)
  </li>
</xsl:template>
</xsl:transform>
```

```
<xsl:template match="person">
```

This tag is just as simple. It says, “When you come across a `<person>` element, put the content in an `` element.” Obviously, this is because we want a list of `<person>`’s.

```
<xsl:template match="person[(@type='chair')]">
```

I put this in just to show that the `match` attribute can be a bit fancier. It is almost the same as the `<xsl:template match="person">` element, but it only applies to `<person>` elements that have an attribute of `type` that equals `chair`. In this way, the chair of the department can be treated differently.

Of course, `<person type="chair">` is also a normal `<person>` element. So how does XSLT know which template to use? The general answer is it picks the most specific. Which is almost always what you want.

Modifying the Processing Order

Now for the hard part, and the piece that makes XSLT so different from CSS. XSLT does not go through the input XML file in linear order. The actions one can take inside a template can affect the *order* of processing. This is crucial and makes XSLT extremely powerful. In fact, XSLT only processes the first input element by default. The overall order of processing is determined by the actions inside the templates.

```
<xsl:template match="department">
```

Consider this element. True enough, when the parser finds the `<department>` element, it outputs some HTML. However, when it gets

to the `apply-templates` element

```
<xsl:apply-templates select="person">
```

things change. This says, “Stop outputting HTML temporarily. Go find all the `<person>` elements in the input, regardless of where they are in the input; for each one you find, find the applicable template and apply that template now.” And the `<xsl:sort>` part says, “Don’t take the `<person>` elements in order that you find them, but rather process them in order of `<name>`, alphabetically.”

Processing the XML with the XSL Rules

So now we run the XML file in figure 1 through an XML processor (we have an XML interpreter on icarus; it’s called SAXON) with the XSLT file in figure 2, and it produces the HTML file in figure 3.

Want to try it yourself? On icarus, copy the XML file (figure 1) into a file called `sample.xml` and the XSL file (figure 2) into `sample.xsl`. (You can cut-and-paste from this newsletter on the Web at <http://www.accc.uic.edu/newsletter/adn28/>.) Then enter the command:

```
saxon sample.xml sample.xsl > sample.html
```

to produce the output HTML file in figure 3.

SAXON is actually a collection of XML tools, including a standard XSLT processor, a Java library, which supports a similar processing model to XSLT but allows full programming capability, and a version of the Ælfred XML parser from Microstar.

The `saxon` command on icarus is a script that executes a piece of the SAXON Java library; to see what it really does, on icarus, enter:

```
more /usr/local/bin/saxon
```

Would you rather work on your personal computer? Try Instant SAXON, an XSLT interpreter for Windows.

The SAXON source and documentation, including explanations of the XSLT tags and lots of examples, are on the SAXON home page at:

```
http://users.iclway.co.uk/mhkay/saxon/
```

In the End

This article is only a small taste of XSLT. There are several good tutorials in book form and on the Web; see “Web Resources” on page 4.

Figure 3: The HTML Output File

```
<HTML>
  <BODY>
    <h1>
      Department of Magical Implications
    </h1>
    <UL>
      <li>Albus Dumbledore
        (<b>Chair</b>)
      </li>
      <li>Minerva McGonagall</li>
      <li>Severus Snape</li>
    </UL>
  </BODY>
</HTML>
```

SOAP: XML Under the Covers

News on the Net

If you think XML is just a new, improved HTML, think again. Related protocols such as [SOAP](#), [Simple Object Access Protocol](#), make it a whole lot more. SOAP is a new way for programs to communicate with other programs, using XML. Normal humans like us won't see this type of XML rendered in any way, but SOAP will make it much easier for programmers to write applications that we humans can use.

Programmers know the value of subroutines. They're the only sensible way to deal with medium-to-large programs: chop them into pieces, with each one encapsulating a different, distinct bit of behavior. Then to perform the overall task, you call the subroutines, or have them call each other, in the right order.

Wouldn't it be particularly clever if we could get our subroutines to run on different computers? This is a networked world, after all, and it makes sense to harness many computers at once. Sometimes this is done simply to increase CPU power. But in most cases, it's done because you want to obtain or change some information that resides on the remote computer.

The process of using remote subroutines as if they were local is called [RPC](#), [Remote Procedure Call](#). Mind you, there are a lot of difficulties making RPC work well. Sending and retrieving data across a network, when the remote machine may or may not be up or responding, is tricky. But the overall concept has a lot of followers, each with their own standards—[DCE RPC](#), [CORBA](#), [RMI](#), and [DCOM](#), just to name a few. And now, SOAP.

Why SOAP, when there already are so many other standards to choose from? In a word, S is for Simple. Each of the other protocols does a lot more and carries a lot more baggage. If you just want the basic functions, independent of language and independent of operating system, with a minimum of runtime services, SOAP may be the best call.

SOAP uses XML to encapsulate the data that needs to be sent to the remote subroutine. And, of course, XML is used to return data from the subroutine and to return notification of any error condition that might have occurred. Any mode of transport can be

WWW Everyone

used for SOAP calls, although HTTP will probably be the most popular.

That's right, the same HTTP that a Web browser uses to talk to a Web server. After all, HTTP can transmit Web pages, even through firewalls, and XML pretty much looks like a Web page, even if the tags and content have a completely different meaning. That's one sign of a well-designed protocol—it can be used cleanly in new circumstances.

This article is not a SOAP tutorial, just an example of how XML can be used to transfer data from one program to another. Suppose I manage twenty workstations and I need to monitor the status of the Web server I'm running on each of them. (Is the Web server up? Heavily loaded? Down?)

Instead of logging onto each one in turn, I put a SOAP server on each (which is not much more than a standard Web server and a CGI script), then I have a central machine send a SOAP request to each one in turn and compile their answers into a report. The SOAP message might be as simple as:

```
<SOAP:Envelope>
  <Getstatus>
    <Process>httpd</Process>
  </Getstatus>
</SOAP:Envelope>
```

I've left off some small details related to XML namespaces, but the basic idea is clear: I want to execute the remote subroutine named `<Getstatus>` and pass it a single parameter whose name is `<Process>` and whose value is `httpd`—send me the status of the HTTPD process that's running on the machine.

Once the SOAP server is ready, it's easy for anyone else to write their own SOAP clients for their own purposes. They can write their processes in perl, C++, Java, COM, JavaScript, or Python, and run them on just about any machine. This simplicity and interoperability make this a very appealing scheme.

SOAP is by no means the only way to transfer data via XML. There are quite a number of protocols under development, many driven by the needs of

business-to-business e-commerce. XML is used because of the ease of interoperability it provides.

For more about SOAP, see:

<http://www.soap-wrc.com/webservices/default.asp>

ebXML, **electronic business XML**, is a more ambitious, business-oriented XML transport protocol; it's at: <http://www.ebxml.com/>

And there are many more.

RSS: Spreading the News

News on the Net

WWW Everyone

Are you drowning in news? There is so much of it on the Web; the sheer volume makes it hard to keep up. Perhaps you have three favorite Web pages for international news, others for local, maybe five or ten for technical updates, a couple for business or

stock quotes, and still others you check for entertainment. Who has time keep track of them all?

How would you like to have your own single customized Web page that combines the headlines from all your favorite sources? Netscape thought you might, and provided just that. Go to My Netscape at <http://my.netscape.com> to set yours up. On the flip side, suppose you were a news producer, would you want your headlines to be available to Netscape's users? That would be easy, too. The magic tool that makes this possible is an XML tag set called **RSS**, **Rich Site Summary**.

There are other RSS services, too. For example, My.Userland.Com, at <http://my.userland.com> (for general news; Userland codeveloped the RSS standard with Netscape), and Meerkat, at <http://www.oreillynet.com/meerkat> (for news with a technical bent). To get an idea of how versatile RSS can be, check them all out. These three sites all use RSS under the covers, but they definitely all have a different look-and-feel.

(Don't like how these sites look? RSS is simple enough that you don't need to use any of them as an intermediary. You can roll your own if you want.)

How RSS Works

The gist of RSS is quite simple. Consider, for example, My Netscape. Each site that wants Netscape to promote its news makes an RSS file. Let's consider the ACCC, for example. If we wanted to make the news items on the ACCC home page available to users of my.netscape.com, we'd start with an RSS file, something like the file in figure 4. The RSS file begins with a small description of the overall site and has a title, description, and link for each news item. That's pretty much all there is to it.

After we made the RSS file in figure 4 available on the Web, we'd register its URL with Netscape or Userland or Oreillynet. After that, Netscape or Userland or Oreillynet would pick up

Figure 4: Sample RSS File

If we at the ACCC wanted the news section on our home page included in RSS news sites such as My Netscape, we would start with an RSS file something like this. The `<channel>` and `<image>` tags describe our overall site and the `<item>` tags mark the individual news items.

```
<?xml version="1.0"?>
<rss version="0.91">
<channel>
  <title>ACCC at UIC</title>
  <link>http://www.uic.edu/depts/ACCC/</link>
  <language>en-us</language>
  <description>The ACCC Web site</description>
</channel>
<image>
  <title>ACCC</title>
  <url>http://www.uic.edu/logos/ACCC.gif</url>
  <link>http://www.uic.edu/</link>
  <width>88</width>
  <height>31</height>
  <description>News about the ACCC and more ...
  </description>
</image>
<item>
  <title>Email filters</title>
  <link>http://www.uic.edu/depts/accc/index.html/news.current.html#0</link>
  <description>There's a new Web email tool on the Mailtools Page, one that automates the creation of filters for incoming email.
  </description>
</item>
<item>
  <title>Mail Relays and Unregistered Machines</title>
  <link>http://www.uic.edu/depts/accc/index.html/news.current.html#1</link>
  <description>When we turned off open mail relays in the ACCC SMTP servers, people with on-campus machines that were not registered in the campus DNS server were unable to send mail.
  </description>
</item>
</rss>
```

our RSS file periodically and display it (with links back to our site) to the users who ask for it.

Syndicated news, simple, yet powerful. Netscape, Userland, and Oreillynet let the user choose which

RSS files they want to view, although I suppose they didn't have to. The key point is that sites can cooperate by sharing data in a standard XML format.

The ACCC Post

Continued from Page 1

New NSKit

There's a new Network Services Kit out, and it's one you should look into, even if you've already got the NSKit or its components installed. This NSKit has a new install feature; it can check on the Web to see whether there's new software to include or updated versions of the software that's already there. (It's similar to Netscape's SmartUpdate.)

Try it out; it might be the last NSKit you have to install from the CD!

New Dialin Lines

To go with the new NSKit, there's a new group of dialin lines, at 1 (312) 666-2001. Like the other ACCC dialin lines, login is required and there's a five-hour time limit. But these lines are different, too—they are set up specifically to be used with the Windows version of the new NSKit, so they use the standard Windows dialin procedure to initiate connections.

(Not installing the new NSKit? See "ACCC Dialin Services" for links to instructions on how to reconfigure your Windows personal computer for these lines; click the [Connect - Home](#) link on the ACCC home page: <http://www.accc.uic.edu>)

New Way of Printing: U-Print

Are you tired of going to the printer in an ACCC lab, sifting through the output, and not finding yours? Are you frustrated because it's not printed yet and you don't know when it will be? Are you upset when you print a job and then find out that the printer in the lab is out of service? Are you concerned about the privacy of your output?

If you are, you're not alone! We have listened to you and are implementing a new way of printing in the ACCC public labs that addresses many of these problems. The U-Print system will make it easier and more reliable to print, will reduce the amount

of wasted output, and will reduce the wear-and-tear on the lab printers.

The key difference in U-Print is that jobs you send to a printer in an ACCC lab will not print until you go to the printer, log in, and select the job to be printed. This happens regardless of where you submit the job from—from a Windows PC or Mac in that lab, from a PC or Mac in another ACCC lab, or from your icarus or tigger UNIX workstation account.

- ✓ The U-Print system will make it easier for you to find your print job output because the job will not be printed until you request it.
- ✓ You will be at the printer ready to pick up your output as soon as it prints, so your privacy isn't compromised.
- ✓ Once your job is submitted for printing, you can print it in any ACCC lab. (This is true in the majority of cases, but there are exceptions. Jobs formatted for output on a specialized printer, such as the plotter in the Art and Architecture lab, must be printed on a compatible printer. Which, in the case of the A&A plotter, is just that plotter.)

Quotas and Charges

Among the main goals of U-Print is to reduce waste, and there's lots of that in the labs. Allowing you to cancel jobs you don't want printed is one way that U-Print addresses this problem, but there's another. We will institute a printing quota, which will not be enforced until the spring semester.

When implemented, the quota will be 150 pages per semester, per person. "Pages" means physical pages printed on black and white printers; duplex pages—ones printed on both sides of the paper—will be charged as one page. Pages printed in color or on other specialized printers may have different rate schedules; the rates will be posted by each printer. (We're experimenting with color printers now and hope to have them available in several labs soon.)

If you go over your quota, you'll be charged 10¢ per physical page printed on black and white printers. Again, other printers may have a different rate schedule.

Oh, and we need to define semester, don't we? The semester is defined as the period between the week before classes start until the week before the next semester starts. Quotas will *not* carry over from semester to semester.

How U-Print Works

1. You submit something to be printed, either from an ACCC public PC or when you're logged in to your tigger or icarus account.
2. When you're ready to print your print job, at any time up to 12 hours after you submitted it, go to a U-Print station next to a printer in an ACCC lab and bring your UIC i-card with you.
3. Insert your UIC i-card in the card reader next to the U-Print station. The card reader works like

an ATM card reader; it holds onto your i-card while you're using the U-Print station.

The card reader front panel will display the amount of money that you currently have on the i-card. For fall semester and while you're below your quota in spring semester and beyond, a total value of \$0.00 is just fine.

4. Login using your ACCC netid and password, select the job you want to print, and click **Print**. (Or **Delete** to cancel the job.)
5. Logout. The card reader will eject your i-card; be sure to take it with you when you go.

For more information, including how you'll add value to your i-card once we begin enforcing the print quota, visit the U-Print Web page:

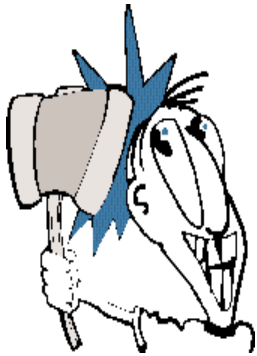
<http://www.accc.uic.edu/plabs/u-print/>

Eudora vs. a Roaming Laptop

The Head Crash



Everyone



Question: I do all of my computing using a laptop that I take from home to office and use on the road. My Internet connection at home comes from a commercial ISP. Because UIC no longer will relay email messages I send from home, I have to tell Eudora to use the correct SMTP server each time I log on from a new place. Thus, at my office I have to configure Eudora to use SMTPSERVX.CC.UIC.EDU but when I go home I have to change it to my ISP's SMTP server. Is there a better way?

Answer: I have a couple of thoughts. The first is to create a copy of your EUDORA.INI file in your Eudora folder; call it, say, EUDORA.ISP. Use a text editor to change:

```
SMTPServer=smtpserv1.cc.uic.edu
```

in that file to your ISP's SMTP server's name.

Then create a new shortcut on your desktop named **Eudora/ISP**. (On Windows, right-click on a blank spot on the desktop, select **New**, then **Shortcut**.)

Use this command line:

```
"C:\Program Files\UICNSKit\Eudora\Eudora.exe" eudora.isp
```

(All on one line and substitute the name of your Eudora folder if it's different from the one above.)

After that, just use the new shortcut to open Eudora whenever you're using your ISP. But

beware that your EUDORA.INI and EUDORA.ISP files will probably get out of sync after a while.

Another thought is to try using Eudora "personalities," each with its own SMTP Server setting. I'm guessing that you can use identical settings from your dominant personality, except the SMTP server. Then when you want to send a message, just double-click the appropriate personality and start typing. For more information on using alternate personalities in Eudora (including with Eudora for the Mac), see:

<http://www.accc.uic.edu/software/eudora/eudalt/>

The real solution is the new "authenticated SMTP" protocol, which will prompt Eudora for your password when you SEND a message. With authenticated SMTP you'll be able to use the UIC server from any location on the Internet. Eudora and Outlook have support for this protocol.

You may want to check with your ISP to see if they provide authenticated SMTP. If so, use your ISP's server for sending email, even while you are on campus.

Jim O'Leary, joleary@uic.edu
ACCC Operating System Support and
Development

The A3C Connection
Academic Computing and Communications Center (MC 135)
Room 124 Benjamin Goldberg Research Center
1940 West Taylor Street
Chicago, Illinois 60612-7352

About The A3C Connection

The A3C Connection is published four times per year by the UIC Academic Computing and Communications Center and provides news and information about the use of computers, communications, and networking at UIC. It is edited by Judith Grobe Sachs with help from Bill Mayer and the UIC Office of Publications Services.

Distribution of the *A3C Connection* is free to UIC faculty, staff, and students, and to other universities and not-for-profit organizations. To subscribe, send us your name and address—UIC campus address if possible, including your department name and mail code. To cancel your subscription, send us your address label or a copy of all the information on it.

Contact us by electronic mail at connect@uic.edu; by telephone at the Client Service Office, (312) 413-0003; by US Mail at The A3C Connection, ACCC (MC 135), Room 124 Benjamin Goldberg Research Center, University of Illinois at Chicago, 1940 West Taylor Street, Chicago, Illinois 60612-7352; or by fax at (312) 996-6834.

We welcome any comments, suggestions, complaints, or requests you might have concerning the *A3C Connection*.

The Fine Print

The use of trade, firm, or corporation names in this publication is for the information and convenience of the reader. Such use does not constitute an official endorsement or approval by the University of Illinois of any product or service to the exclusion of others that may be suitable. Trade names that may appear in this publication include the following: Apple and Macintosh (trademarks of Apple Computer, Inc.); IBM, VM/CMS, VM/ESA, PC-DOS, and OS/2 (trademarks of International Business Machines Corporation); UNIX (a registered trademark licensed exclusively through X/Open Company, Ltd.); HP, HP-UX, SPP-UX, HP-Convex, and Exemplar (trademarks of Hewlett-Packard Corporation); Sun (a registered trademark of Sun Microsystems, Inc.); and Microsoft, MS-DOS, Windows, Windows NT and Windows95 (trademarks of Microsoft Corporation). All other product names mentioned herein are used for identification purposes only and may be the trademarks or registered trademarks of their respective companies.

Permission is granted to reprint or adapt all or part of the *A3C Connection* for nonprofit use, provided that full acknowledgment of the source is given.