10 mistakes *you* can made with XML and Perl

“more bugs, more quickly”

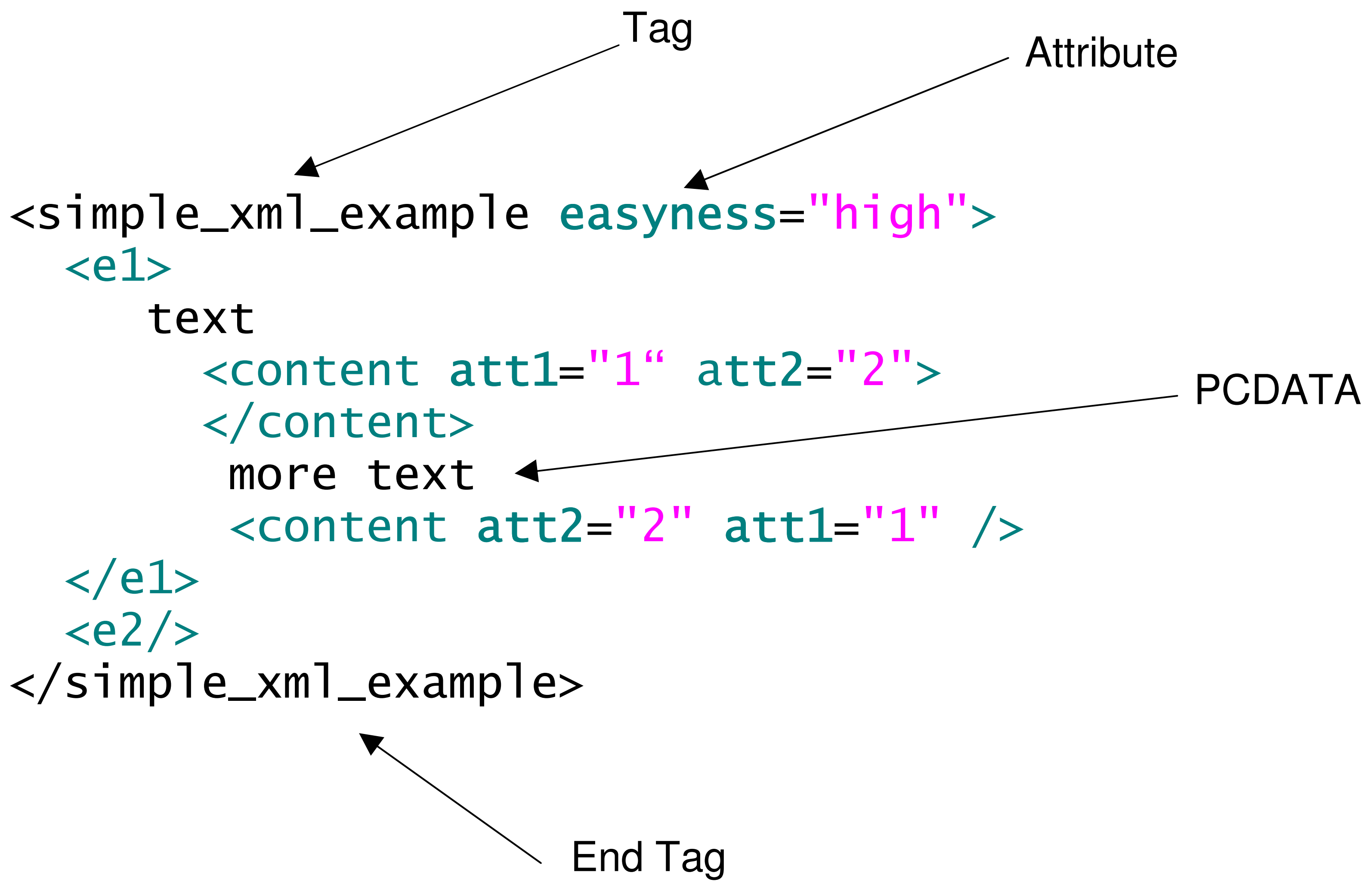
Andy Adler

Remember: *It’s all fun and games …*
until someone laughs their head off.

What is XML?

- Andy's Answer:
  - ASCII syntax for data structures
- XML looks like HTML
  - However, conceptually very different
- Fairly simple spec
  - “Tons” of associated specs
- Standards mostly controlled by fairly sane people
  - Broad support from open source to MS

Simple XML example



```
<simple_xml_example easyness="high">
  <e1>
     text
       <content att1="1“ att2="2">
       </content>
        more text
        <content att2="2" att1="1" />
  </e1>
  <e2/>
</simple_xml_example>
```

More compilated example

- File format for StarOffice 6.0 (ie openoffice.org) is XML based
- Text sections represented in XML Images are referenced. File is zipped.
- Used the following document:

```
 <H1>
Just another OpenOffice.org
hacker
</H1>
```

More complicated example

Processing Instruction

Document Type Declaration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE office:document-content PUBLIC
  "-//OpenOffice.org//DTD OfficeDocument 1.0//EN"
  "office.dtd">
<office:document-content
    xmlns:office="http://openoffice.org/2000/office"
    xmlns:text="http://openoffice.org/2000/text"
    xmlns:fo="http://www.w3.org/1999/XSL/Format"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:math="http://www.w3.org/1998/Math/MathML"
    office:class="text"
    office:version="1.0">
    <office:script/>
```

Definition of Namespace “office”

XML Namespaces usage

More compilated example

```
<style:font-decl  style:name="Albany"
                  fo:font-family="Albany"
                  style:font-family-generic="swiss“
                  style:font-pitch="variable"/>
  </office:font-decls>
  <office:automatic-styles/>
  <office:body>
      <text:sequence-decls>
          <text:sequence-decl
              text:display-outline-level="0"
              text:name="Text"/>
      </text:sequence-decls>
      <text:h  text:style-name="Heading 1"
               text:level="1">
          Just another OpenOffice.org hacker
      </text:h>
  </office:body>
</office:document-content>
```

XML colouring courtesy of vim

This is too easy …

- I promised that I'd help you write buggy XML code
- But this looks ***way*** too easy.
- How can we write
  - Hard-to-debug
  - Job-security-enhancing
  - Happen-only-occasionally

  bugs with this stuff?

  Just wait, my friends …

Bug #1: Confusing Attributes and Data

- Some people get worried when they need to create a new XML format to encode their data:

- Should I put the information in
  - ☐ attributes

  or

  - ☐ text (PCData)?

Bug #1: Confusing Attributes and Data

- Should I say:

```
<sandwich>
    <ingredient name="marmite“
                action="spread"/>
</sandwich>
```

Or

```
<sandwich>
     <ingredient>
          <name>marmite</name>
          <action>spread</action>
     </ingredient>
</sandwich>
```

Bug #1: Confusing Attributes and Data

- Answer:

Both

- In fact, alternate between them.
- After all, *there's more than one way to do it*

Bug #2: Abusing bloated and buggy standards

- Example #1: SOAP:

```
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
     "http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:execute xmlns:ns1="urn:facerec-service“
      SOAP-ENV:encodingStyle=
         "http://schemas.xmlsoap.org/soap/encoding/">
      <command-list xmlns:ns2=
         "http://schemas.xmlsoap.org/soap/encoding/"
               xsi:type="ns2:Array"
               xmlns:ns3 = "urn:USER_NAME_SPACE"
               ns2:arrayType="ns3:Command[10]">
        <item xsi:type="ns3:OUR_DATA_HERE">
```

Finally, Something We care about

Bug #2: Abusing bloated and buggy standards

- Example #2: MathML:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
   <mi>F</mi>
     <mrow>   <mo>(</mo>
              <mi>y</mi>
              <mo>)</mo>
     </mrow>
     <mo>=</mo>
     <mfrac>
        <mn>1</mn>
        <mrow>
          <mn>2</mn>
          <mi>&pi;</mi>
        </mrow>
     </mfrac>
```

$$F(y) = \frac{1}{2\pi}\int_{-\infty}^{\sqrt{y}} \left(\sum_{k=1}^{\pi} \sin^2 x_k(t) \right) f(t)\,\mathrm{dt}$$

All this XML gets us to the fraction

Bug #2: Abusing bloated and buggy standards

- Lots more examples:
  - XML Schema
  - RDF
  - Open office format

- Unfortunately, some aren't too bad
  - SVG *<= from Adobe no less!*

Luckily, these are few

Bug #2: Abusing bloated and buggy standards

- There are 2 ways to parse these bloated documents
  - Reach into the data structure and grab the elements you care about

  Or

  - Carefully parse the structure and check everything

- The best part is: **They’re both wrong!**

Approach #1: only grab the elements you care about

- Consider this document

```
<recipe for="coffee">
   <ingredient name="coffee beans"/>
</recipe>
```

- I parse with "/recipe/ingredient[1]@name" This is XPath syntax → *later*
- Now, we get the following document instead:

```
<recipe for="chile">
   <ingredient name="halepeño beans"/>
</recipe>
```

Approach #2: Carefully parse the structure and check everything

- Our server carefully parses this SOAP message

```
<SOAP-ENV:Envelope xmlns:xsd=
     "http://www.w3.org/1999/XMLSchema">
```

- Then one day, our application breaks, after *lots* of debugging, the client now sends:

```
<SOAP-ENV:Envelope  xmlns:xsd=
     "http://www.w3.org/2001/XMLSchema">
```

- How the *&^%^& is our application supposed to know that the 2001 and 1999 schema specs are essentially identical?

Bug #3:
Abusing bloated and buggy tools

- The basic XML API is the Document Object Model (DOM)
- Here's a sample

```
my $nodes = $doc->
        getElementsByTagName ("CODEBASE");
my $n = $nodes->getLength;
for (my $i = 0; $i < $n; $i++)   {
    my $node = $nodes->item ($i);
    my $href = $node->
        getAttributeNode ("HREF");
    print $href->getValue . "\n";
}
```

What’s wrong with DOM

- Simple answer:

It goes against everything Perl stands for

What’s wrong with DOM

- Lets compare long and painful XML API’s to string parsing in C
- The fact that its painful, tends to encourage people to take shortcuts for ***micro-optimization***
- Example:
  - `/* Remove .txt extension from file name */`
  - `filename[ strlen(filename)-4 ] =0;`
- Correct Solution:
  - `$filename =~ s{ \.txt $}{}x;`

What's wrong with DOM

- This kind of shortcut is:
  - Hard to read
  - Hard to write
  - Tends to result in other bugs
  - Leaks of pointers, variables
- And worst of all
  - When your concentration is forced onto painful code, you loose

cognitive momentum^{TM}

Bug #4: How to confuse validity and well-formedness

- Naturally, one would expect that a “valid” document conforms to the specification.

Wrong!

- A document that conforms to the specification is called “**well-formed**”
- A “valid” document conforms to its DTD
  - or maybe its schema,
  - or maybe its RELAX spec (or any one of the many ways to specify the document syntax)

How is this a bug?

- Valid XML documents are rare
- Although in theory, DTD’s etc. are a good idea, I’ve rarely seen them in practice.
  Reasons include:
  - Industry time pressures
  - General sloppy thinking about testing
  - Lack of tool support
  - Specification overload: DTD’s were painful, but the W3C replacement is worse
  - Important specs like SOAP don’t support validation
- Therefore:

Almost any XML document is **invalid**

More about validity

- Here's some interesting information about SOAP

```
"Roger Wolter[MS]" <rwolteronline@microsoft.com>
  wrote in message> news:...
  >This quote from the SOAP standard may
  >clarify things:
  >>A SOAP message MUST NOT contain a Document
  >>Type Declaration. A SOAP message
  >>MUST NOT contain Processing Instructions.
```

- Therefore SOAP requires **invalid XML**

Bug #5: Unreliable parsing with regexps

- You may think that there should be some good packages out there to parse XML

You're right

- There are **lots** of API's to parse XML:
    - DOM
    - SAX
    - JDOM (Java DOM)

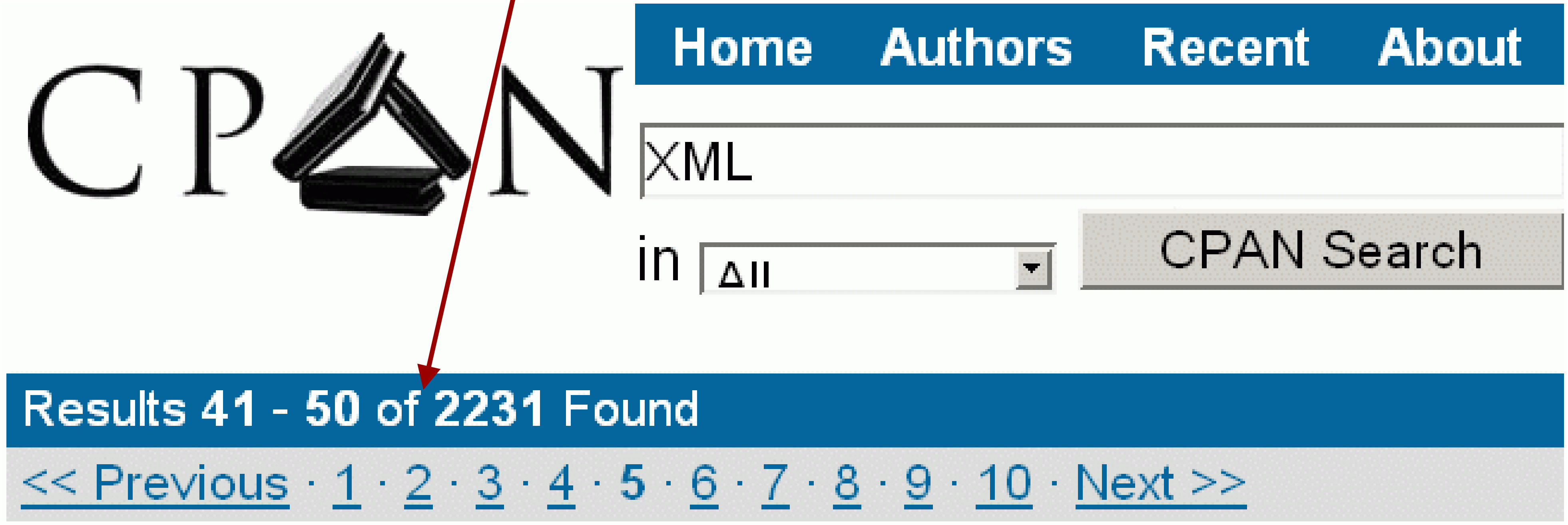
Bug #5: What about perl API's?

- There are **lots and lots** of perl API's to parse XML:

XML::SAX::PurePerl	XML::Dumper	XML::Simple
XML::Config	XML::Handler	XML::LibXML
XML::Twig	XML::Filter::Digest	DBIx::XML
XML::GDOME	XML::Mini	XML::SAX::Base
XML::SAX::Expat	XML::Ximple	XML::YYLex
XML::FOAF	XML::Grove::Subst	XML::Template
XML::DOM	PApp::XML	Meta::Lang::Xml
XML::RAX	XML::Generator	XML::Checker
RPC::XML::Parse	XML::Xerces	XML::ValidWriter

Bug #5: What about Perl API's?

There are **lots and lots and lots** of perl API's to parse XML:


Home
Authors
Recent
About
XML
in
All
CPAN Search
<< Previous · 1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10 · Next >>

Results 41 - 50 of 2231 Found

Bug #5: Unreliable parsing with regexps

- Many people have thrown in the towel and said *&^%$ it.
  - XML is text
  - Perl parses text
  - Therefore: I’ll use a regular expression
- Even Tim Bray (a founder of XML) admits to this
  - www.tbray.org/ongoing/When/200x/2003/03/16/XML-Prog

Bug #5: Unreliable parsing with regexps

- However, parsing balanced text with regexps is (deliciously) unreliable
- Consider:

```
<sandwich>
    <ingreds name="marmite"
          action="spread"/>
 </sandwich>
```

- Parsing:

```
/ingreds name="[\"]*" action="[\"]*"/;
$name = $1; $action = $2;
```

- Problem: attribute order is not part of XML infoset

Bug #6: Unreliable encoding with printf

- Example: encode passport data in XML
- Code:

```
printf “<passport>%s</passport>”,$data;
```

- Problem: my passport data is

```
$data=“CANADLER<<ANDREW<<<<“;
```

- So now XML is:

```
<passport>
   CANADLER<<ANDREW<<<<
</passsport>
```

Bug #6: Unreliable encoding with printf

Unfortunately, there are good ways to get around it in perl

- Quote your <&> tags:

```
s/</\&lt;/g
```

- Use a module:
  - XML::Twig
  - XML::Writer
- Or use CDATA

```
print “<tag><[!CDATA[“ .
      $data . “]]></tag>”;
```

Bug #7: Parsing to make your code crawl

■ You say:		use XML::DOM
	or	use XML::Simple
■ which says:		use XML::Parser
■ which says:		use Expat
■ which uses:		expat.xs
■ which parses the text		

Bug #7: Parsing to make your code crawl

- Expat is a standard API for lots and lots of XML code: Perl, Apache, Python, Mozilla
- Its fast and well understood
  - History: Created by James Clark, a British *expat* living in Thailand
- Expat has some weirdnesses
  - It doesn’t validate
  - API calls don’t necessarily return full data elements
  - Statically linked with patches to many modules
- Using XML::Parser (or better expat.pm) directly is a great source of code pain

Bug #8: Parsing to make your memory thrash

- Take this presentation, convert to openoffice.org format.

```
$ unzip yapc.ca-XMLtalk.sti content.xml

$ ls -l yapc.ca-XMLtalk.sti content.xml
  291358 content.xml
   47124 yapc.ca-XMLtalk.sti
```

- Now, lets parse this with XML::Simple

```
$ perl -MXML::Simple
     -e'$t=`cat content.xml`;'
     -e'$r=XMLin($t);';
```

- Size of $r is 4656k (cygwin perl, win2k)

zipped data =	1
XML format =	6.2 times bigger
Perl objects =	100 times bigger

Bug #8: Parsing to make your memory thrash

- A common complaint is that XML is bloated data format.

It's true

- However, compressing XML is good
  - Often *.xml.bz2 is smaller than an equivalent bespoke binary format
- The real bloat happens in the object representation from the parser

Bug #9: How to abuse broken name spaces support

- Name space support is kind of like using a local variable in perl

```
    local $soap= …
    $soap->{command}= …
```

- XML Example

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/“>
   <SOAP-ENV:Body> ...
```

- The use of “SOAP-ENV”, should be an arbitrary (if usual) choice

Bug #9: How to abuse broken name spaces support

- Fortunately, for very many implementations (especially of SOAP), the namespace is hardcoded
- This means that you can have:
  - XML documents that meet the spec that don’t work with the parser
  - XML documents that don’t meet the spec (ie. have the namespace point elsewhere) and are still accepted and processed

Bug #10: How to modify your XML during validation

- Here’s an underexploited bug: Did you know that the “validation” process is allowed to change the XML document?
- The DTD (or schema) is allowed to specify default attiributes that are inserted into the document during validation

Bonus Bug #1: Store information outside the infoset

Information that is not in the *infoset* should not make any difference

- Character encoding
  - Example: é vs. &eacute;
- PCDATA vs CDATA
- Whitespace in elements
  - Example <foo att="bar"> vs. <foo   att = "bar" >
- Order of attributes
  - Example <foo att1="bar1" att2="bar2"> <foo att2="bar2" att1="bar1">

Bonus Bug #1: Store information outside the infoset

Non infoset information, *cont'd*

- Entities vs PCDATA
- Namespace names
  - SOAP Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/“>
   <SOAP-ENV:Body> ...
```

versus

```
<ROPE-ENV:Envelope xmlns:ROPE-ENV=
  "http://schemas.xmlsoap.org/soap/envelope/“>
   <ROPE-ENV:Body> ...
```

Bonus Bug #2: Calling. MethodsWithNamesThatAreWayToo LongAndMakeCodeHardToRead();

I call this effect JAVA itis

Bonus Bug #3: Backwards thinking with callbacks

Warning:

code guru's, plug ears now.

Bonus Bug #3: Backwards thinking with callbacks

- Most of the sophisticated XML API's require you to use callbacks
- Example (with XML::Parser, others are worse)

```
use XML::Parser;
$p=new XML::Parser(Handlers => {
   Start => \&handle_start,
   End =>   \&handle_end,
   Char =>  \&handle_char});
$p->parse('<foo id="me">HW</foo>');
```

Bonus Bug #3: Backwards thinking with callbacks

- Now we just need to write callback functions

```
sub handle_start {
  # Step 1:get input parameters
  # Step 2:figure out where we are
}
```

- Keeping track of state in the callback function is a real pain.
  - We don’t have a proper object to store it
  - Tend to use hacks like static variables
  - Makes us do unnecessary mental gymnastics

Bonus Bug #4: Broken tools

- MSXML didn't work at all well until version 3. It still doesn't handle external entities properly
- No Debuggers for XML parsers.
  - Now activestate has a visual XSLT
- No good tools for writing DTDs and schemas. So most people don't bother

Bonus Bug #4: Broken tools in perl

- Lets try to install SOAP::Lite from CPAN
  - Just to be clear SOAP::Lite is a great module
- Look at prerequisites
- `SOAP::Lite`
  - `MIME::Lite`
    - `MIME::Parser`
      - `MIME::Tools`
        - **IO::Stringy**
        - **Mail::Field**
        - **Mail::Header**
        - **Mail::Internet**
  - `XML::Parser`
  - **HTTP::Daemon**
  - **LWP::UserAgent**
  - **URI**

$$\left.\vphantom{\begin{matrix}a\\b\\c\\d\\e\\f\\g\end{matrix}}\right\} \quad et \quad cetera$$

Actually, there's lots good about XML. *slashdot.org* quote:

Re:But XML is great for computers... (Score:5, Insightful) by Ed Avis (5917) <ed@membled.com> on Tuesday March 18, @08:48AM (#5535893)

You mean like most other non-xml config files in /etc, like say hosts, DNS zone files, named.conf, passwd/shadow, hosts.allow/deny, sendmail.mc or resolv.conf (etc. etc.)? These have standard layouts, text-based, can be edited by hand and can be easily parsed.

You just gave the best argument for adopting XML as widely as possible. Yes, all these can be parsed (with the possible exception of sendmail's config files which may be Turing-complete) but they all require ***different*** code for each config file. If they were in XML you'd still need different semantic code, of course, but a whole wodge of syntax issues (how do I quote strings, how do I escape newlines, how do I mark nested scopes, what happens when the string delimiter character occurs inside a string, how do I deal with comments, what is the character set, is there a formal grammar for the document, etc etc) would be dealt with. … But they would be dealt with ***once***. No need to learn a new or almost-the-same-but-slightly-different set of syntactic conventions for every single config file.

Non Bug #1: XML is text

- Unfortunately, XML doesn’t have a bizarre binary encoding like ASN-1 (or msword).
- Even worse, XML has a consistent, clean implementation of Unicode UTF-8.
  - Unlike what JAVA or MS say, Unicode is not a 2 byte character, that’s the BMP-1 subset of UTF-16
  - UTF-8 is compatible with ASCII, and allows C style strings (with null termination)
- Your best bet for bugs is to badly integrate broken unicode support in JAVA
  - Unfortunately, this talk is about Perl

Non Bug #2: clean separation of syntax and semantics

- Unfortunately, XML is now quite well understood to be a data structure
  - Some think it's a programming language
  - Some think it's HTML
- Rigid definition of well-formedness
  - Unlike pdf, where Adobe writes a spec, but actually parses documents differently
  - Style sheets allow conversion of XML data to different presentation formats (and are slow, slow, slow)

XML Advice

- Don't hand parse XML
- Be careful hand creating XML
  - need to “quote”, < , > , &
- Don't use XML::DOM
  Don't use XML::Parser

Advice: use XML::Simple

- Use XML::Simple

```
$xml_struct = XMLin(
        xml file or string )
# gives a hash of arrays data structure
$xml_string = XMLout ( $xml_structure )
```

- Good for small XML documents.
- Lots of options to control how structure is created. *I don't like the default options*

Advice: use XML::XPath::Simple

- Xpath is like a *regexp language for XML*

```
use XML::XPath::Simple;
$xp = new XML::XPath::Simple(
   xml => $xmlstring,
   context => '/' );
$content = $xp->valueof(
         '/doc/c[2]/d[1]@id');
```

Future of XML and Perl

- XML will not be well used until there is good in-language support *(à la Perl regexps)*
- It must be easier to use XML than to invent an *ini style file format
- Require “under the covers” caching
- Perl XML support is too scattered -we need less and better modules.
- The Perl way is/should be to make it easy to right thing, the Java way is to forbid the wrong thing. We must perlize XML.

Ten Mistakes with XML and Perl, Andy Adler, YAPC::CA, Ottawa, May 15, 2003

Abstract: XML will enable universal data interoperability by providing a vendor independent, OSI level 7 presentation layer, semantic capable, interoperable data format. Since XML is so easy to use, it is unclear why there is a need for talks on how to use it. Fortunately, with the power of Perl, and a few helpful tips, anyone can add hard to track bugs to their XML handling code. We will focus on the following techniques:

- Confusing Attributes and Data
- Abusing bloated and buggy standards
- Abusing bloated and buggy tools
- How to confuse validity and well-formedness
- Unreliable parsing with regexps
- Unreliable encoding with printf
- Parsing to make your code crawl
- Parsing to make your memory thrash
- How to abuse broken name spaces support
- How to modify your XML during validation

Now, new and improved, including 4 bonus bugs.