XML Tools in Perl

Geir Aalberg
geir@aalberg.com

Nordic Perl Workshop 2006

Myths about XML

- “Unicode with pointy brackets”
- Too hard to parse
- All data must be put inside CDATA blocks
- Namespaces don't work
- XSLT will never take off
- What's wrong with using Perl data structures?

What is XML?

- A syntax
  - Simplified SGML, much easier to parse
- A data structure
  - tree-based, cross.platform
  - industry standard tools
- A technology family
  - SAX, DOM, XPath, XSLT, XQuery
  - XHTML, WML, RDF/XML, RSS/Atom, SOAP, ODF, ebXML

The 10 XML Commandments

1. Thou shalt think of XML as a tree structure, not as a string

History of XML

- Generalized Markup Language (GML)
  - 1969: Invented by Goldfarb, Mosher and Lorie at IBM
  - Over 90% of all IBM documents produced using GML
- Simple Generalized Markup Language (SGML)
  - 1980: First draft by ANSI
  - 1986: ISO standard 8879
  - Major users include US DoD, AAP
  - 1988-96: DSSSL developed into ISO 10179
  - 1991: O'Reilly and HaL Computer Systems design DocBook
  - 1992: Tim Berners-Lee designs HTML

History of XML

- Extensible Markup Language (XML)
  - 1996: XML Working Group
  - 1998: XML 1.0 W3C Recommendation
  - 1998: DOM W3C Recommendation
  - 1999: XSLT and XPath W3C Recommendations
  - 2000: XHTML 1.0 W3C Recommendation
  - 2001: XML Schema W3C Recommendation
  - 2001: RELAX NG OASIS spec + part of ISO 19757
  - 2006: XQuery W3C Recommendation Candidate

The 10 XML Commandments

2. Thou shalt not make unto thee any illegal markup

XML Syntax

- Wellformed (legal) XML
  - correctly nested opening and closing tags

```
<foo><bar><baz/></bar></foo>
```

  - [&<>”] must be encoded as entities (or CDATA)

```
&amp; &lt; &gt; &quot;
```

  - parsing non-wellformed documents *must* cause fatal error
- Encoding
  - ASCII, ISO-8859-1 or (default) UTF-8
  - Always UTF-8 internally

The 10 XML Commandments

3. Thou shalt not XML namespaces in vain

Namespaces

- Motivation
  - To avoid tag name collisions
  - To allow processor handlers in pipeline (e.g. XSLT)
- Namespace determined by scope
  - much like Perl
- Namespace is empty string unless stated otherwise
  - Common pitfall when using XPath
- The prefix is irrelevant after parsing
  - Only the tag name and namespace URI counts

Namespace prefixes

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <body>
        <p>This page intentionally left blank.</p>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

is the same as

```
<stylesheet version="1.0"
  xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="/">
    <html:html xmlns:html="http://www.w3.org/1999/xhtml">
      <html:body>
        <html:p>This page intentionally left blank.</html:p>
      </html:body>
    </html:html>
  </template>
</stylesheet>
```

Namespace prefixes

Or indeed

```
<stylesheet version="1.0"
  xmlns="http://www.w3.org/1999/XSL/Transform">
  <template match="/">
    <html xmlns="http://www.w3.org/1999/xhtml">
      <body>
        <p>This page intentionally left blank.</p>
      </body>
    </html>
  </template>
</stylesheet>
```

- These are all exactly similar!
  - Try transforming any XML document with them and load into Firefox. Use `.xhtml` extension to force correct MIME type (`application/xhtml+xml`).

The 10 XML Commandments

4. Honor thy DTD and XML Schemas: that thy working days may be short upon the land which the BOSS giveth thee

Validation

- DTD
  - Legacy from SGML
  - Does not follow XML syntax (but can be included inline)
  - Does not understand namespaces
  - Can define entities (unlike schemas)
- XML Schema
  - Schema used by W3C
- RELAX NG
  - Schema used by most others
  - Both XML and simpler non-XML syntax

DTD/Schema generators

- Very useful as a starting point
  - DTD syntax is pretty arcane and hard to remember
- Generate when needed
  - may catch typos that will take a long time to debug
- Online tools
  - http://www.hitsw.com/xml_utilites/

The 10 XML Commandments

5. Thou shalt cache thy DTDs and Schemas locally to avoid unneccesary HTTP requests

XML Catalogs (libxml example)

- Local repository of DTD/Schemas
  - Resolves official URIs to local files

```
$ cat /etc/xml/catalog
   <?xml version="1.0"?>
   <!DOCTYPE catalog PUBLIC "-//OASIS//DTD XML Catalogs V1.0//EN"
   "file:///usr/share/xml/schema/xml-core/catalog.dtd">
   <catalog xmlns="urn:oasis:names:tc:entity:xmlns:xml:catalog">
   <delegatePublic publicIdStartString="-//Norman Walsh//DTD Slides"
     catalog="file:///etc/xml/docbook-slides.xml"/>
   <delegateSystem systemIdStartString="http://docbook.org/xml/"
     catalog="file:///etc/xml/docbook-xml.xml"/>...
```

- Use `xmlcatalog` tool to add/remove

```
$ xmlcatalog /etc/xml/catalog "-//W3C//DTD XHTML 1.0 Transitional//EN"
file:///usr/share/xml/xhtml/schema/dtd/1.0/xhtml1-transitional.dtd
```

The 10 XML Commandments

6. Thou shalt not parse thy XML with regular expressions

Processing XML

- Stream-based parsing
  - SAX
- Tree-based parsing
  - DOM
  - XPath
  - XQuery
- Convert to Perl structure

Streaming vs tree-based parsing

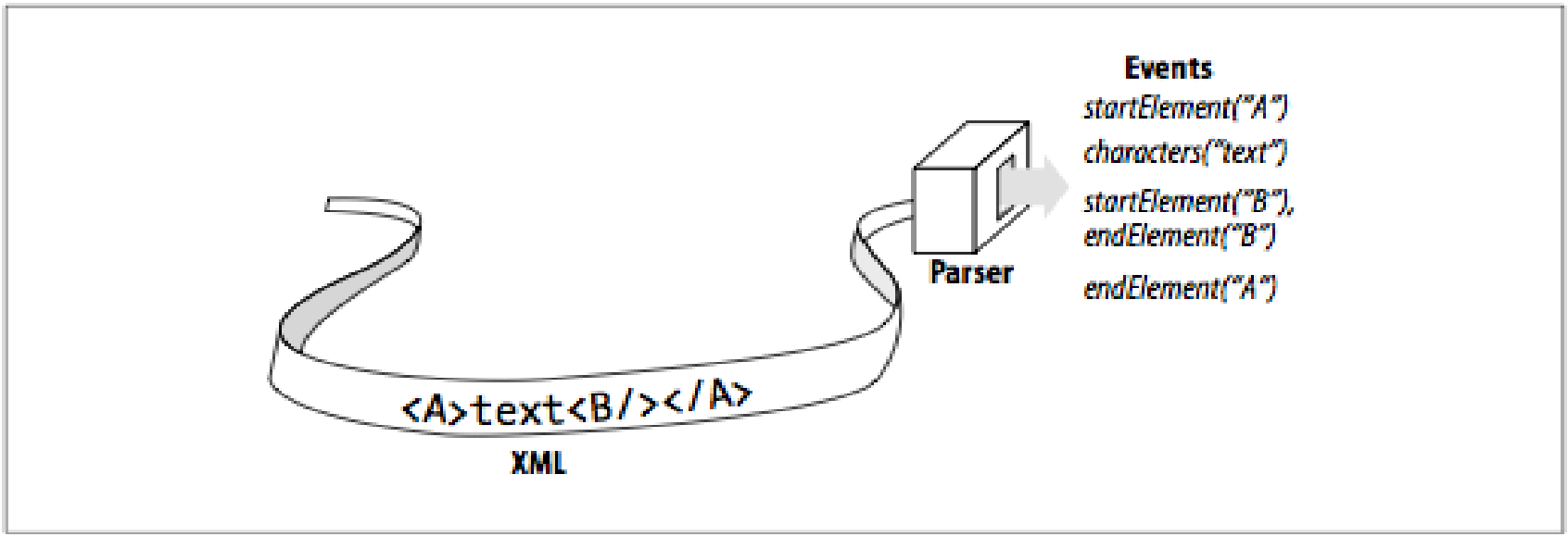


Figure 1-1. A streaming parser API

Figure 1-2. An object tree parser API

Simple API for XML (SAX)

- Stream-based parsing
- Emphasis on *simple*
- Suitable for large documents
- Event handlers for each node (start, content, end)
- No way to backtrack/lookahead
- Namespace support from v.2 (SAX2)

Document Object Model (DOM)

- Cross-platform API for processing XML tree
  - Same in Perl, C, Java, Javascript et al
  - Familiar to AJAX programmers
- Set of standard methods

```
getElementById()
setAttribute()
createElement()
replaceChild()
```

- DOM Level 2 adds namespace support
- verbose compared to XPath and XSLT

XPath

- Developed in conjuction with XSLT spec
- Functional query language

```
/html/body/div[@class="sect"]/h1[count(following-sibling())>1]
```

- One line XPath = 10 lines of Perl

DOM/XPath example (Javascript)

```
var rightcol = document.evaluate("/html/body/table[7]", document,
null, XPathResult.FIRST_ORDERED_NODE_TYPE, null).singleNodeValue;

var mybox = document.evaluate("/html/body/table[7]/tbody/tr[1]/td[6]",
document, null, XPathResult.FIRST_ORDERED_NODE_TYPE,
null).singleNodeValue;

if (rightcol) {
    var holder = rightcol.parentNode;

    if (mybox) {
        var gone = mybox.parentNode.removeChild(mybox);
        var newtable = document.createElement("table");
        holder.appendChild(newtable);
        var newtr = document.createElement("tr");
        newtable.appendChild(newtr);
        newtr.appendChild(gone);
    }

    holder.replaceChild(newtable, rightcol);
}
```

Same example in XSLT

```
<xsl:template match="/html/body/table[7]">
    <table>
        <tr>
            <xsl:copy-of select="tbody/tr[1]/td[6]"/>
        </tr>
    </table>
</xsl:template>
```

XQuery

- Similar to SQL, but for XML trees instead of tables

```
for $b in $books/book[price < 100]
order by $b/title
return $b
```

- Not yet an official W3C Recommendation
- Few tools support it yet

The 10 XML Commandments

7. Thou shalt choose thy XML parser wisely

XML parser libraries

- James Clark's expat
  - C. Non-standard, stream-based API (but not SAX)
- GNOME libxml
  - C, with OO Perl bindings
- Apache Xerces
  - C++ (Also Java)
- Platform specific
  - .NET (MSXML), Apple Cocoa NSXML
- Pure Perl

Parser features

	expat	libxml	Xerces	.NET	NSXML
DTD validation	N	Y	Y	Y	
XML Schema	N	Y	Y	Y?	
RELAX NG	N	Y	N	?	
Namespaces	?	Y	Y		
SAX2	N	Y	Y		
DOM	N	Y	Y		
XPath 1.0	?	Y			
XPath 2.0	N	?		N	N
XQuery	N	N			Partly
gzip	N	Y			

command line tools

- xmlwf (expat)
  - check for wellformedness
- xml_pp (XML::Twig)
  - code reformatter
- xmllint (libxml)
  - check/validate documents

```
--format      # code reformat/indent
--compress    # output gzip data
--xinclude    # process XIncludes
--valid       # validate before XInclude
--postvalid   # validate XIncluded document
--shell       # this is cool!
```

XML::Parser

- Perl granddaddy of XML
- Based on James Clark's expat
- Non-standard API
- Expects string input, returns string output
- Not suitable for pipeline processing

XML::SAX

```
use XML::SAX;
use MySAXHandler;

my $parser = XML::SAX::ParserFactory->parser(
  Handler => MySAXHandler->new
);

$parser->parse_uri("foo.xml");

package MySAXHandler;
use base qw(XML::SAX::Base);
sub start_document {
  my ($self, $doc) = @_;
  # process document start event
}

sub start_element {
  my ($self, $el) = @_;
  # process element start event
}
```

XML::Twig

- SAX-like interface on top of expat
- Discards nodes after use, suitable for large files

```
my $twig=XML::Twig->new(
  twig_handlers =>
    { title   => sub { $_->set_tag( 'h2') }, # change title tags to h2
      para    => sub { $_->set_tag( 'p')  }, # change para to p
      hidden  => sub { $_->delete;        },  # remove hidden elements
      list    => \&my_list_process,           # process list elements
      div     => sub { $_[0]->flush;      },  # output and free memory
    },
  pretty_print => 'indented',                 # output formatted
  empty_tags   => 'html',                     # outputs <empty_tag />
);
```

XML::LibXML

- Implements SAX, DOM, XPath (but not XQuery)
- Faster and more robust than anything else
- Plugins for XUpdate
- Mix and match DOM, XPath and XSLT on same tree
- Works hand-in-hand with XML::LibXSLT and other libxml-based modules

XML::LibXML example

```
use XML::LibXML;

my $parser = XML::LibXML->new();
my $tree = $parser->parse_file('text.xhtml');
my $root = $tree->getDocumentElement;

foreach ($root->findnodes('/html/body/div[@class=”sect”]')) {
  printf “%s (%s chars)\n”,
    $_->findvalue('h1[1]'),
     length($_->findvalue('.'));
}
```

XML::XPath

- More unwieldy than XML::LibXML

```
use XML::XPath;
use XML::XPath::XMLParser;

my $xp = XML::XPath->new(filename => 'test.xhtml');

my $nodeset = $xp->find('/html/body/div[@class="sect"]/h1');

foreach ($nodeset->get_nodelist) {
    printf "%s\n", XML::XPath::XMLParser::as_string($_);
}
```

- `xpath` utility can be handy for debugging

```
$ xpath transitional.html '/html/head/title/text()'
Found 1 nodes:
-- NODE --
Quick Example
$
```

XML::Xerces

- Little or no Perl documentation
  - See C++ API at Apache site

Pure Perl parsers

- XML::SAX::PurePerl
  - From author: “XML::SAX::PurePerl is slow. Very slow. I suggest you use something else in fact.”
- XML::Stream::Parser
  - 50 % slower than XML::Parser
  - Could be useful where installing libraries not possible

The 10 XML Commandments

8. Thou shalt not convert XML to Perl data structures without good reason

XML::Simple

- Convert XML into hashes and arrays

```
<config logdir="/var/log/foo/" debugfile="/tmp/foo.debug">
  <server name="sahara" osname="solaris" osversion="2.6">
    <address>10.0.0.101</address>
    <address>10.0.1.101</address>
  </server>
</config>
```

```
{
    'logdir'        => '/var/log/foo/',
    'debugfile'     => '/tmp/foo.debug',
    'server'        => {
        'sahara'         => {
            'osversion'     => '2.6',
            'osname'        => 'solaris',
            'address'       => [ '10.0.0.101', '10.0.1.101' ]
        }
    }
}
```

XML::Smart

- Similar to XML::Simple
  - each point in the tree work as a hash and an array at the same time
- Caveat
  - Some users report encoding problems

The 10 XML Commandments

9. Thou shalt not generate thy XML output using `print` statements

XML::API

- Uses XML Schema to generate methods
- XHTML API available

```
use XML::API::XHTML;
my $x = new XML::API::XHTML();

$x->head_open();
$x->title('Test Page');
$x->head_close();

$x->body_open();
$x->div_open({id => 'content'});
$x->p('A test paragraph');
$x->div_close();
$x->body_close();

$x->_print;
```

XML::Writer

```
use XML::Writer;
use IO::File;

my $output = new IO::File(">output.xml");

my $writer = new XML::Writer(OUTPUT => $output);
$writer->startTag("greeting",
                  "class" => "simple");
$writer->characters("Hello, world!");
$writer->endTag("greeting");
$writer->end();
$output->close();
```

- **Warning**
  - Does not check for illegal characters; may produce incorrect XML

XML::LibXML::Tools

```
my $dom = $lxt->complex2Dom( data =>
                              [ document =>
                                [ node =>
                                  [ deeper_content =>
                                    [ $tools->attribute("attribute",
                                                        "value"),
                                      "deep content" ],
                                  ],
                                  node => [ "content" ]
                                ]
                              ]
);
```

- This is now ready to process further
  - eg with LibXSLT
- Fails tests... needs more work?

XML::RSS

- Parser/generator
- Supports RSS 0.9, 0.91 and 1.0
- XML::RSS::LibXML recommended
  - easier to extend with own namespaces
  - can be processed further with LibXSLT

XML::PYX

- Use standard UNIX filters on XML

```
$ pyxhtml dirty.html | pyxw > clean.html
```

- Can clean up “dirty” HTML


```
$ pyxhtml dirty.html | pyxw > clean.html
```

XML::XSH

- Shell for working inside XML documents
  - Similar to `xmllint –shell`
  - Seems to have namespace parsing problems
- Use pipes to add remote functionality

```
xsh> ls DOC:/ | ssh my.remote.org 'cat > test.xml'
```

The 10 XML Commandments

A. Six days shalt thou labour, unless using XSLT

XSLT

- XML::XSLT
  - Perl. Alpha versjon; incomplete. Dead?
- XML::LibXSLT
  - C. Fast (twice as fast as Sablotron). xsltproc
- XML::Sablotron
  - C++
- XML::Xalan
  - Java? Commited to XSLT 2.0. Slower than Saxon

The 10 XML Commandments

B. Thou shalt not make unto thee any more old-style HTML

Why XHTML?

- Faster parsing in browser and spiders
  - Said to improve Google PageRank
- Better suited for mobile devices
  - smaller memory footprint
- It's the future!
- XHTML 2.0 brings cool stuff
  - `<section>` and `<h>` for better structuring
  - any tag can contain `href` and `src`
  - XForms

XHTML requirements

- Must be 100 % legal XML
  - Browsers will croak if illegal

```
<img alt=”Bang &amp; Olufsen 15&quot; speakers”/>
```

- Serve as `application/xhtml+xml`
  - text/html is reserved for SGML
- Use correct DTD and namespace

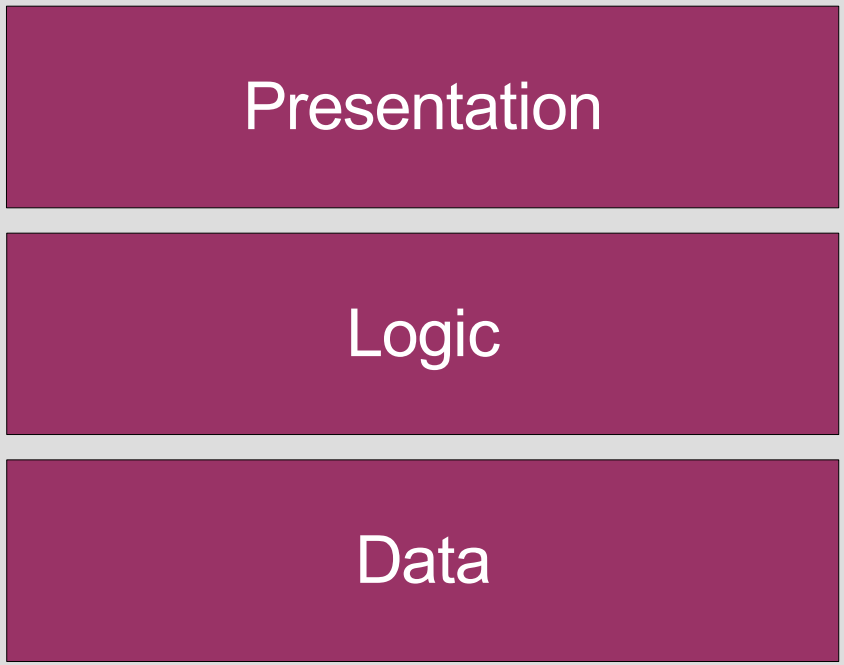
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

Template systems

- Model-View-Controller (applied on web apps)
  - Model = your data
  - View = HTML markup
  - Controller = everything else?
- MVC is only relevant for GUI applications
  - “Controllers contain the interface between their associated models and views and the input devices (e.g., keyboard, pointing device, time).”

http://c2.com/cgi/wiki?WhatsaControllerAnyway

Web application layers



Markup (HTML, WML, RSS)

Perl code

Class::DBI, webservice

Separating logic from presentation

- Hardcoding HTML in Perl

```
print <<EOT
  <p>$name<br/>$address</p>
EOT
```

- Hardcoding Perl in HTML (Mason)

```
<ul>
% foreach $item (@list) {
  <li><% $item %>
% }
</ul>
```

- Both are equally bad
- Neither handles entity encoding

Common template systems

- Must encode entities automatically
- Template Toolkit
  - Template::Plugin::XML (hopefully)
  - Template::Plugin::XML::LibXML (probably)
- HTML::Mason
  - Does not encode; has no grasp of XML
- HTML::Template
  - Ditto

Conclusion

Now you know why
Jesus had 10 disciples