XML Modules

Perl XML modules by example

- XML modules presentation
- Example: problem specification
- Code examples
- Conclusions

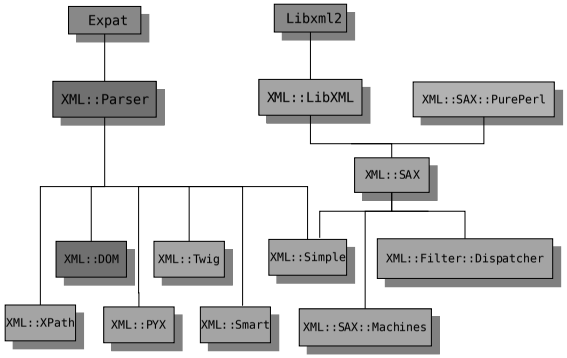
XML::Modules

- 145 XML::.* modules on May 22cd
- How to choose?
  - Application space (data vs document)
  - Size of the data set
  - Processing Model
  - Quality

TLAs and Other Accronyms

- DOM
  - an Object Model and an API for tree processing
- SAX
  - an API for stream processing
- Xpath
  - a query language for XML documents
- XSLT
  - an XML transformation language

XML::Module::Map



Other Useful Modules

- DBD::AnyData
- XML::Generator::DBI
- XML::SAX::Writer

  XML::Handler::YAWriter

  XML-Filter-DataIndenter

Six Blind Men and An Elephant

- XML is big and complex
- “XML is very like a spear”:

  XML invoices
- Definitely data
- Small documents
- Exchanged as XML
- The data should live in a Data Base

Data vs Doc

Text is messy, data is simpler!

- Data has more structure
- Data has no mixed-content
- Some tools work best (or only!) for data-oriented XML
- Most XML these days is data-oriented

Document-oriented XML

```
<para>
  Typically applications will just include the
  <filename>gnome.h</filename> header file, the
  <filename>libgnome.h</filename> or the
  <filename>gnome-i18n.h</filename> header file and this
  will define the <link linkend="gettext-
macro">_</link>()
  macro as an invocation to <link
  linkend="gettext">gettext</link>(). In the case of
  libraries, shared object modules (dynamically linked
  libraries) or shared object CORBA servers (CORBA
servers
  implemented as shared  libraries), you should define
the
  <emphasis>GNOME_EXPLICIT_TRANSLATION_DOMAIN</emphasis>
  macro ...
</para>
```

Data-oriented XML

```
<Finvoice Version="1.0">
 <SellerPartyDetails>
  <SellerPartyIdentifier>0123456-7</SellerPartyIdentifier>
   <SellerOrganisationName>Pullin Kala Oy</SellerOrganisationName>
   <SellerOrganisationTaxCode>0123456-7</SellerOrganisationTaxCode>
   <SellerPostalAddressDetails>
    <SellerStreetName>Haapatie 7</SellerStreetName>
    <SellerTownName>Helsinki</SellerTownName>
    <SellerPostCodeIdentifier>00100</SellerPostCodeIdentifier>
   </SellerPostalAddressDetails>
 </SellerPartyDetails>
 <InvoiceDetails>
  <InvoiceNumber>1/2002</InvoiceNumber>
  <InvoiceTotalVatExcluded="EUR">100,00</InvoiceTotalVatExcluded>
  <InvoiceTotalVat Currency="EUR">22,00</InvoiceTotalVat>
  <InvoiceTotalVatIncluded Currency="EUR">122,00</InvoiceTotalVatIncluded>
 </InvoiceDetails>
</Finvoice>
```

Typical data-oriented XML

- Standard Documents
- Configuration files
- Data Base dumps
- Serialized objects
- XML-RPC, SOAP messages

Typical use of data-oriented XML

XML is an EXCHANGE format

- Extract data
  - Use Perl data structures
  - Put it in a Data Base
- Add data
- Convert the XML document
  - From external to internal DTD
- Avoid complex XML to XML transformations

Finvoice

- The electronic invoice of the Finnish Bankers' Association
- An exchange format for invoices
- Sent by Seller to Buyer
- http://www.fba.fi/finvoice/index.html
- Other standards for invoices: Visa, SAP

Specification of the Problem

```
Receive an invoice
Check it (PO #, qty...)
  If OK
    store the relevant data in a DB
  Else
    # bonus credits only
    add error diagnostic to the
     document
    output the updated XML
```

TIMTOWDI

- Get the data and forget about the XML
- Load the XML and process it (tree-mode processing)
- Process the XML as it is being parsed (stream-mode processing)
- Still more ways...

Load'n Forget

- XML::Simple, XML::Smart
- Map the XML into Perl native data structures
- Content of an element:

```
$xml->{BuyerPartyDetails}->{BuyerOrganisationName}
```

- Value of an attribute:

```
$xml->{BuyerPartyDetails}->{BuyerOrganisationName}
```

- Repeated Elements:

```
my @rows= @{$xml->{InvoiceRow}};
```

Tree Processing

- XML::LibXML, XML::XPath, XML::DOM, XML::Twig
- Load the XML into a tree
- Use navigation or queries to access data
- Navigation

```
my $child= $node->getFirstChild;
my @nodes= $node->getElementsByTagName( 'InvoiceRow');
```

- Queries (Xpath)

```
$doc->findnode( '/Finvoice/InvoiceDetails');
```

Stream Processing

- XML::PYX, XML::SAX, XML::SAX::*
- Check and fill a Perl data structure
- 2 passes
- Use chained SAX filters to separate checking and loading the data

TIMT3WTDI

- DBD::AnyData
- Regexps (shame on you!)
- XSLT

Other issues

- Validation
- Encodings

Conclusion

- Data-oriented XML can be very easy to process
- Very little XML-specific code
- Love your trees
- Have Fun!

Resources

- The Perl-XML FAQ

  `http://perl-xml.sourceforge.net/faq/`
- XML.com Perl-XML articles

  http://xml.com/pub/q/perlxml
- The perl-xml mailing list

  `http://listserv.activestate.com/mailman/listinfo/perl-xml`
- The XML Cover Pages

  http://www.oasis-open.org/cover/sgml-xml.html
- The annotated XML specification

  http://xml.com/axml/axml.html
- Zvon.org

  http://zvon.org