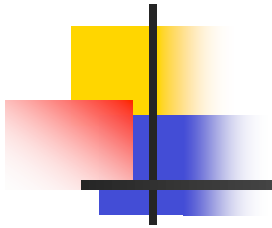
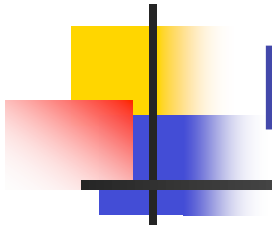


B336 Internet Systems Programming

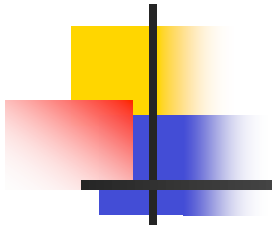
XML Parsing and Processing

(Week 7 Lecture 2)



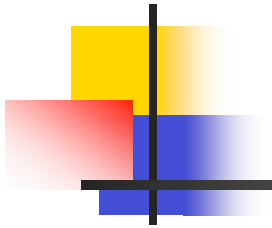
Learning Objectives

- Learn some of the available modules in Perl to parse and process XML documents.
- Look at some example code in using these modules.



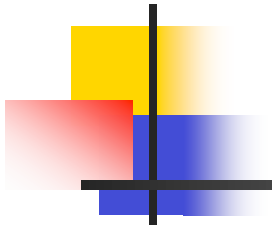
Learning Objectives

- In the scheme of what we are doing in this unit:
  - We are studying how to use XML as an important set of Internet technologies to use as solutions in different areas.
  - A major part of developing an XML solution is to design and implement software to deal with the information in these XML documents. To do this, we must know how to parse and process XML documents.
  - Perl has a set of modules for XML. We will use that as an introduction to tools available in a programming language to handle XML.



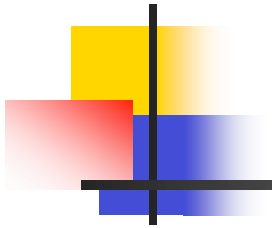
Lecture Outline

- The XML::Parser module
- The XML::DOM module
- Other modules in Perl



Parsing XML in Perl

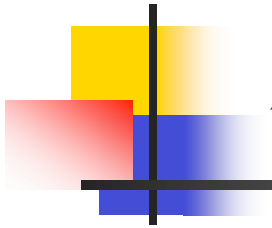
- As mentioned in the previous lectures, before your program is able to do anything with the data in an XML document, it must first
  - parse the document and extract the relevant data
  - put the relevant data in appropriate data structure for program to use
- In Perl, we can use the XML::Parser module to do this.



XML::Parser

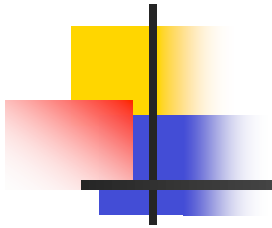
- XML::Parser is a basic Perl module we can use to parse an XML document.

- It is based on James Clark's `Expat` library.
  - Expat is a very important C library used extensively to implement XML parsers.
  - Most of the low level details of Expat is hidden in parser such as XML::Parser.



XML::Parser

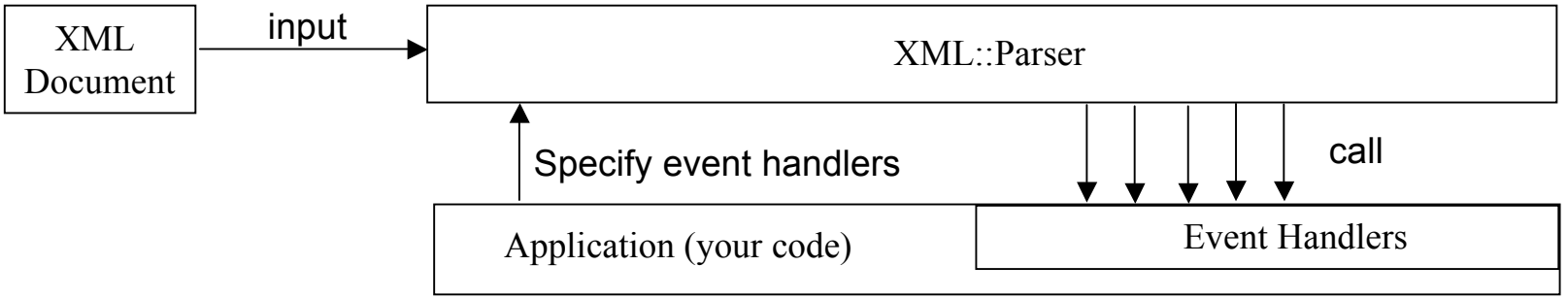
- Expat and XML::Parser are non-validating parsers.



Event handlers in XML::Parser

- XML::Parser is event-driven.
  - It works by allowing you to specify event handlers for different situations the Parser encounters during parsing.
  - You do not have to deal with the low-level character-by-character parsing (and verifying). You only what happens when you encounter certain “components” of an XML document (eg. a start tag).

XML::Parser Processing



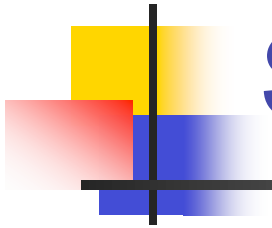
An Example using XML::Parser

```
use XML::Parser ;

my @tags ;

my $parser = new XML::Parser ;
$parser->setHandlers (Start => \&my_start_handler) ;

die "Unable to parse XML document\n"
    unless $parser->parsefile("poetry.xml") ;

# Do something with @tags
# ...


sub my_start_handler
{
    my ($expat, $item, %attr) = @_ ;
    print "Encountered node type: $item \n" ;
    push (@tags, $item) ;
}
```

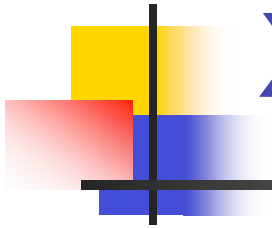
Everytime encountering a start tag, call subroutine &my_start_handler.

&my_start_handler will take node-type value from @_, print a message, and put the node-type into the @tags array.

XML::Parser and Data Structures

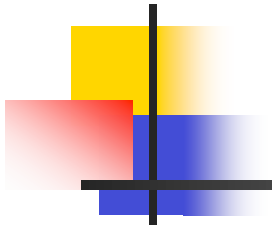


- XML::Parser does not construct a useful object or data structures to represent the XML document.
  - That is left to the programmer using XML::Parser.
  - When using XML::Parser, we think about the events that occur rather than the objects to be manipulate.



XML::DOM

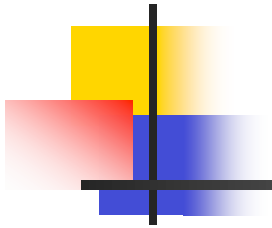
- XML::DOM is an alternative module to XML::Parser that can construct a useful data structure to manipulate.



In this unit...

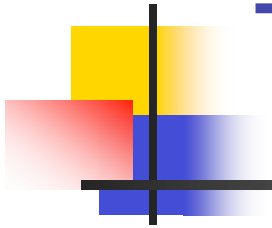
- In the labs and assignment for this unit, we will concentrate on using XML::DOM instead of XML::Parser.

The Document Object Model (DOM)



- The DOM is a platform- and language-independent describing the content, structure and style of documents, by putting them in specified *objects*.

- DOM is currently a W3C standard
  - Currently developing level 3
  - Based originally on Netscape's, and later Microsoft's, efforts to allow client-side scripting (eg JavaScript) to manipulate HTML pages.



The DOM Structure Model

- The DOM presents documents as a hierarchy of ***node*** objects, some with ***child*** nodes of various types, and others with ***leaf*** nodes that cannot have anything below them.
  - This basic idea is the basis behind the structure of XML documents.
  - XPath is build on the same concept.

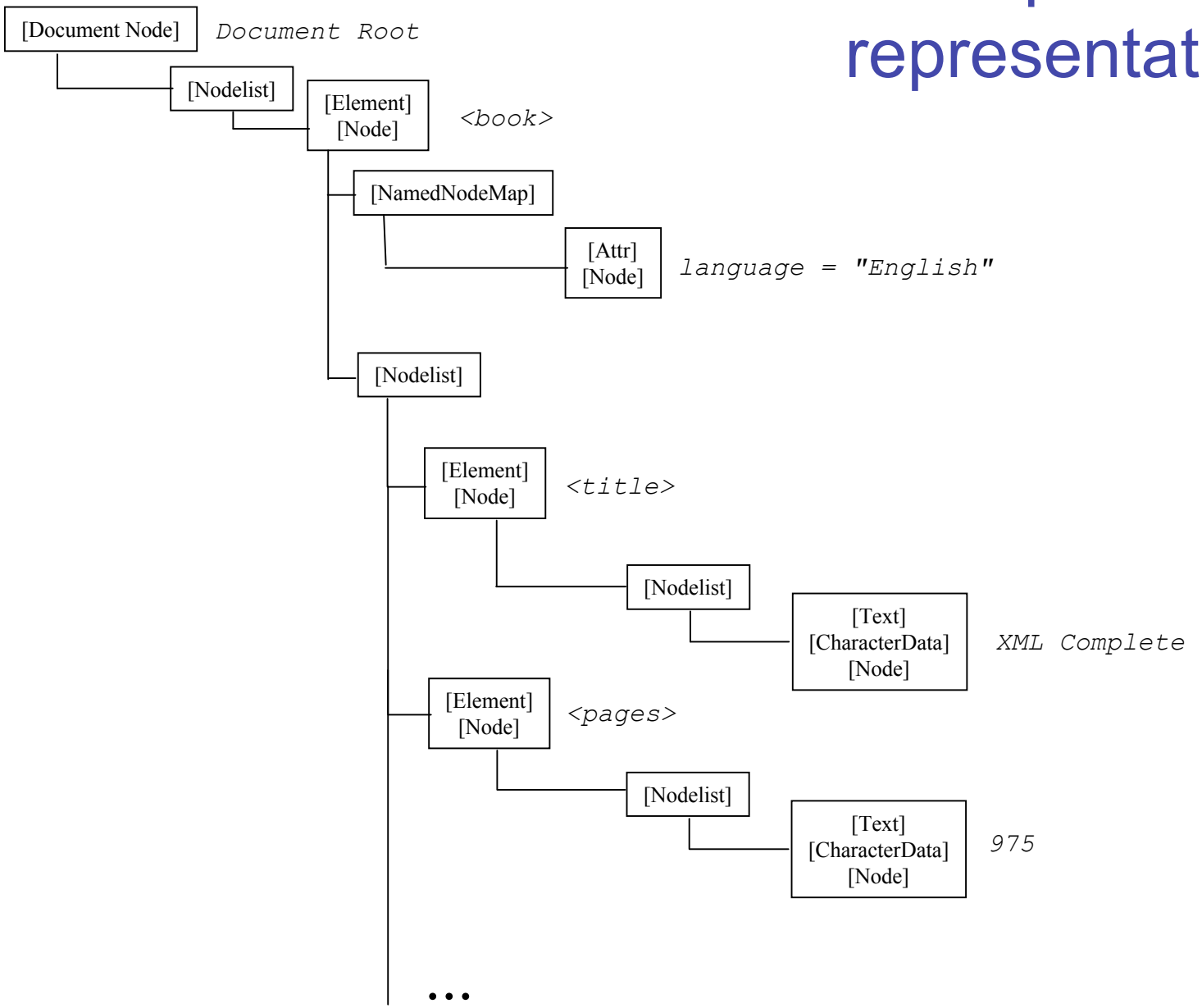
The DOM Structure Model

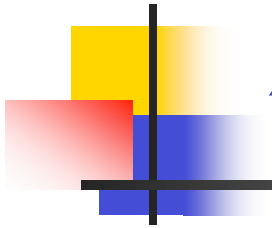
- Some example DOM objects:
  - **`Document`** - consisting of **`Element`**, **`ProcessingInstruction`**, **`Comment`**, **`DocumentType`**
  - **`Element`** - consisting of **`Element`**, **`Text`**, **`Comment`**, **`ProcessingInstruction`**, **`CDATASection`**, **`EntityReference`**.
- For a full set of DOM level 1 objects, see:
  - http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

Example XML Document

```
<?xml version="1.0"?>
<book language="English">
    <title>XML Complete</title>
    <pages>975</pages>
    <publisher>
        <name>Sybex</name>
        <city>San Francisco</city>
    </publisher>
</book>
```

Example XML::DOM representation

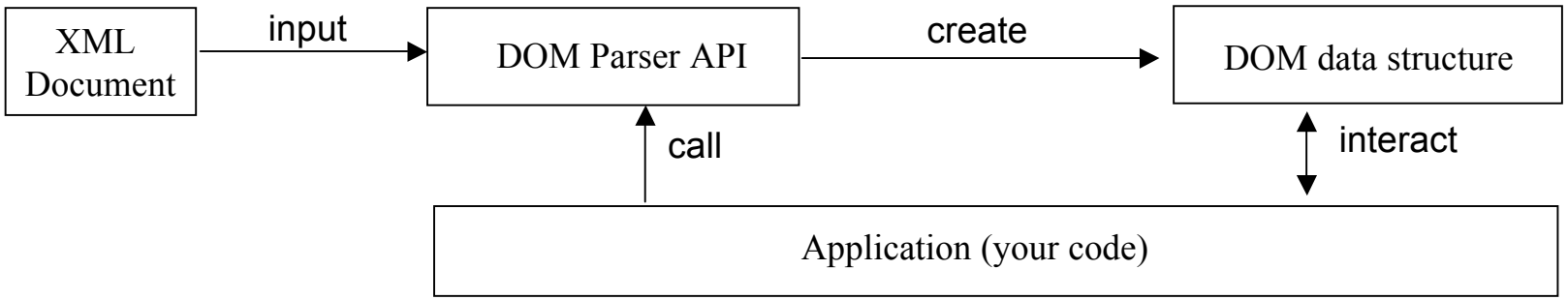




XML::DOM

- XML::DOM is a DOM level 1 compliant parser module in Perl
  - It extends XML::Parser by constructing a DOM object when parsing.
  - The programmer can then manipulate this object after calling the parsing method.

XML::DOM Processing

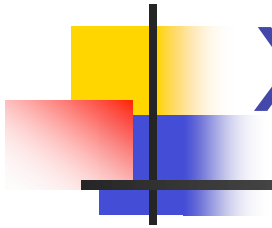


Node Types in XML::DOM

- XML::DOM has the following predefined constants to refer to types of nodes:

Constant	Actual Value
UNKNOWN_NODE	0
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3
CDATA_SECTION_NODE	4
...	
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
...	
XML_DECL_NODE	15
ATTLIST_DECL_NODE	16

Some example subclasses of XML::DOM



- XML::DOM::Node
  - class describing any node in the DOM structure (ie. the XML tree)
- XML::DOM::NodeList
  - class describing a collection of nodes.
- XML::DOM::Element
  - class describing any element
- XML::DOM::Attr
  - class describing any attribute
- XML::DOM::Text
  - class describing text nodes (these nodes contains the text between the start and end tags).

Example Methods in XML::DOM::Node

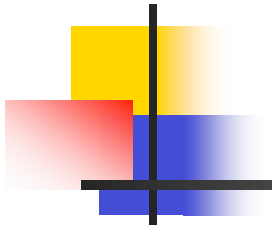
- Access data in the DOM object:
  - **`getNodeType`**
    - Returns an integer indicating the type of node (see list on page 21)
  - **`getNodeName`**
    - Returns the name of the node as a string.
  - **`getChildNodes`**
    - Returns a list of all the children of the node.
  - **`getFirstChild`**
    - Returns the first child of the node as an object.
  - **`getLastChild`**
    - Returns the last child of the node as an object.
  - **`hasChildNodes`**
    - Returns true if the node has child nodes.

Example Methods in XML::DOM::Node

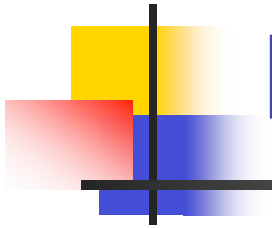
- **`getAttributes`**
  - Returns an object containing all the attributes of the node.
- **`getElementsByTagName("string")`**
  - Returns a list of all the elements under the node with the name “string”

- Change the DOM object:
  - **`insertBefore (newnode, referencenode)`**
    - Insert the newnode immediately before the referencenode
  - **`replaceChild (newnode, oldnode)`**
    - Replace oldnode with newnode
  - **`removeChild (childnode)`**
    - Delete childnode from the tree
  - **`appendChild(childnode)`**
    - append childnode to the end of the node’s children.

Example Methods in XML::DOM::NodeList



- **`item(i)`**
  - Returns the content of the i^{th} item in the list of nodes. Index starts from 0.
- **`getLength`**
  - Returns the number of items in the list.

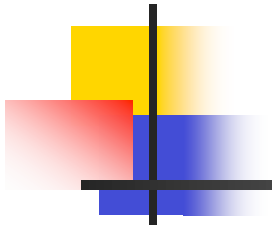


Methods in XML::DOM::Element

- Inherits from XML::DOM::Node, so has all the methods in XML::DOM::Node.
- Most of the nodes in a DOM tree are of this type.

Methods in XML::DOM::Element

- Example extra methods:
  - **`getTagName`**
    - Returns name of the element as a string.
  - **`setTagName(newname)`**
    - Change the name of the element.
  - **`getAttribute(attributename)`**
    - Returns the value of the attribute as a string.
  - **`setAttribute(attributename, value)`**
    - Gives an attribute a new value.
  - **`removeAttribute(attributename)`**
    - Remove an attribute from the element node.

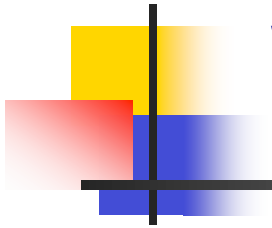


Methods in XML::DOM::Text

- The second most common node type in a DOM tree (after XML::DOM::Element)
- Example extra methods:
  - **`getData`**
    - Returns text of the node
    - Can also use **`getNodeValue`**
  - **`setData(text)`**
    - Set the data in the node to the text given.

A Simple Example using XML::DOM

```
use XML::DOM ;

my $parser = new XML::DOM::Parser ;
my $dom_obj ;

die "Unable to parse XML document\n"
    unless $dom_obj = $parser->parsefile("course.xml") ;

my @nodes = $dom_obj->getElementsByTagName("unit") ;

foreach $elem (@nodes) {

    if ($elem->getNodeType == ELEMENT_NODE) {
        print $elem->getTagName, "\n" ;
    }

    # Do other things with $elem
    # ...
}
```



What the script does

1. Open course.xml and parse it.

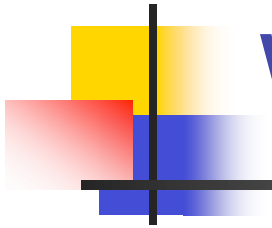
2. If document is not well-formed then stop with an error message.

3. If document is well-formed, then create a DOM object and assign it to `$dom_obj`.

4. Get all elements with name “unit” from `$dom_obj`. For each of those elements:
    a. Print the tag name
    b. Do other things...

Another Example using XML::DOM

```
use XML::DOM;
my $doc ;

my $xml_file= shift ;

my $parser = new XML::DOM::Parser;
die "Unable to parse XML document\n"
    unless $doc = $parser->parsefile ($xml_file);

foreach $elem ($doc->getElementsByTagName ("title")) {
       foreach $child ($elem->getChildNodes) {
              print  $child->getNodeValue ;
       }
       print  "\n" ;
}
```



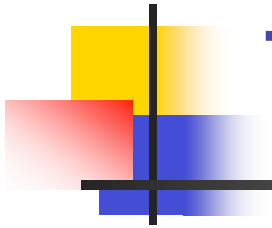
What the script does

1. Reads a file name from the command line. Open and parse the file.

2. If document is not well-formed then stop with an error message.

3. If document is well-formed, then create a DOM object and assign it to `$doc`.

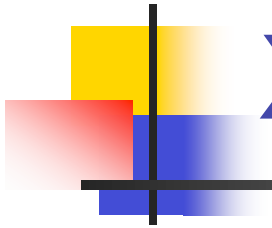
4. Get all “title” elements from `$doc`. For each of them:
    a. Get all its child nodes, ie. the text nodes containing the “title” text, and print the text.



Text nodes in elements

- The text in the tags actually belongs in a child Text node of elements node, instead of the element node itself.
  - Eg. In the previous code, the text in the ”title" tag actually belongs in a child of the ”title" node, instead of the ”title" node itself.
- Refer to the diagram on page 19 again.

Another Example using XML::DOM



- Convert the whole constructed DOM object into a string, and print it.

```
use XML::DOM;

my $xml_file= shift ;

my $parser = new XML::DOM::Parser;
my $dom_obj = $parser->parsefile ($xml_file);
print $dom_obj->toString;
```

Another Example using XML::DOM

- Removes every “surname” child element from every “unit” element.

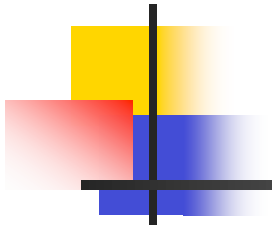
```
use XML::DOM;

my $course_file= shift ;

my $parser = new XML::DOM::Parser;
my $course = $parser->parsefile ($course_file);
my @units = $course->getElementsByTagName("unit") ;
foreach $u (@units) {
        foreach $child ($u->getChildNodes) {
                $u->removeChild($child)
                        if ($child->getNodeName eq "surname") ;
        }
}
print $course->toString ;
```

An Example of other XML modules

```
use XML::XSLT;

my $xsl_file = "default.xsl" ;
my $xml_file = "course.xml" ;

my $xslt = XML::XSLT->new ($xsl_file);

$xslt->transform ($xml_file) ;
print $xslt->asString ;
```

1. Opens and parses an course.xml.
2. Transform course.xml using XSLT templates in default.xsl.
3. Prints the result on screen.

* Note: not all of XSLT is support by the XML::XSLT module. See module documentation on what is supported.



References

- Perl online documentation for XML::DOM.
- For other XML packages, search www.cpan.org.