

XML

Extensible Markup Language

- Used to describe data
 - common file/document structure
 - data sharing among applications
 - document validation
 - parsing libraries
 - generation libraries

Issues

- Syntax
- Document Type Definition/Schema
- Browser Support
 - CSS
 - XSL (Extensible Style Language)
- Parsing
 - Dom (Document Object Model)
 - Sax (Simple API for XML)

XML Syntax

- Tags (like HTML)
 - tag names are not fixed (like in HTML).
 - tag names **are** case sensitive
 - **every** start tag must have a corresponding end tag.
 - can use things like `
`, ``
- Entire document is a single strict hierarchy.
- Attributes (like HTML)
 - attribute values **must be quoted**

[Student Record in XML \(viewable\)](#)

Document *elements*

- An element begins with a start tag and ends with the corresponding end tag.
- The element *contains* everything between the start and end tags
 - text (character data)
 - other elements

```
<foo>  
    Hello  
    <blah>Bye</blah>  
</foo>
```

Tag Names

- Characters can be

- alphabetic
- numeric
- ' '
- _

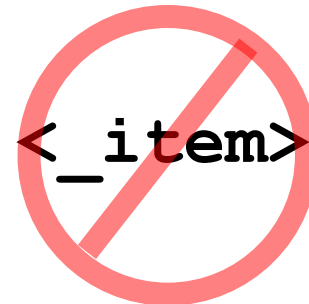
Other punctuation characters are legal, but best to avoid. Never use '-', '.', or ':' !!!

- Must start with alphabetic character

`<foo22>`

`<super_hero>`

`<stolen_song>`



More about Tag Names

- In general: use tag names that are descriptive.
- Never use the tag name 'xml', or in fact, any tag name that starts with 'xml'.
- You usually don't need long tag names
 - the context typically is also descriptive.

```
<student>
  <name>Dave</name>
  <grades>
    <tests>
      <average>99.5</average>
    </tests>
    <hw>
      <average>99.35</average>
    </hw>
  ...
XML
```

Attribute Values must be quoted

```
<student id="660123456">
```

```
<h3 align="center">
```

```
<table border=0 cellspacing=2>
```

```
<stolen_song name="Hey Jude">
```

Reserved Attribute Names

- **xml:lang**: used to specify the language used to represent the text of a tag.

```
<name xml:lang="en">Joe</name>
```

- **xml:space**: used to indicate whether whitespace is significant or not
 - The default is that whitespace is significant
 - This is different than HTML! (whitespace is not significant) in HTML.

Namespaces

- Used to avoid tag name *collisions*.
- tag name is preceded by a namespace:

```
<eiw:student>
```

```
<eiw:grade>
```

```
<eiw:term_project>
```

Document Prolog

- XML Declaration

- top line in any xml document
- indicates the xml version (use 1.0).
- Can also indicate document encoding.

```
<?xml version="1.0">
```

- Document Type Definition

- declares valid tag and attribute names/values
- used for validation
- DTD is optional.

XML Document (Prolog) Example

```
<?xml version="1.0"?>
```

```
<!DOCTYPE student SYSTEM "stu.dtd">
```

```
<student>
```

```
  <name>Dave</name>
```

```
  <grades>
```

```
    <tests>
```

```
      <average>99.9</average>
```

```
    </tests>
```

```
  ...
```

Document Type Definition

- Formal definition of the structure (hierarchy) of a specific kind of XML document.
- List of *legal* elements
- Lists of *legal* attributes.

domain specific information.

- Example: what a *student record* should look like as an XML document.

DTD

- A DTD can be *inline* or stored in an external document.
 - much like you can have a CSS style section in an HTML file, or refer to an external style sheet.

```
<?xml version="1.0">  
<!DOCTYPE student [  
  <!ELEMENT student (name,grades)>  
  <!ELEMENT name (#PCDATA)>  
  <!ELEMENT grades (tests,hw)>  
  ...  
>
```

DTD Syntax

- A DTD is not an XML document
 - there is an alternate format for specifying the same information that is in the form of an XML document
 - XML Schema Definition (XSD)
- DTD comes from SGML, which was around long before XML, HTML, the web, etc.
- Simple syntax, but it's very different than XML.

DTD and XML

- An XML document is made up of:
 - Elements, Tags, Attributes
 - Entities shortcuts, special chars
 - PCDATA parsed character data
 - CDATA character data
- DTD describes how these can be combined to create a specific kind of document.

DTD: Element definition

- Element name (tag that contains the element).
- what sub-elements must be/can be within the element.
- whether sub-elements can appear once, multiple times, etc.
- The order of subelements within the element.
- Whether PCDATA can appear within the element

DTD Element Examples

- `<!ELEMENT student (name,id,email)>`
 - must have these subelements (in order!)
 - `<name>...</name>`
 - `<id>...</id>`
 - `<email>...</email>`
- `<!ELEMENT name (#PCDATA)>`
 - name element must include only text (no subelements).
- `<!ELEMENT id (#PCDATA|ssn)>`
 - id element must contain either text, or a ssn element.

Other ELEMENT definitions

- You can also define elements that can contain
 - optional content (**grades?**) or (**grade***)
 - mixed content (**test|hw|lab**) *
 - one or more instances of the same kind of subelement (**grade+**)
 -
- There are more ways to define elements...

Attributes

- For each element type, you can specify legal attributes and what kind of values each can have.
- There are many *types* an attribute can have (check a DTD reference for the full list).
- The type defines what values the attribute can have
 - could be CDATA (any text)
 - could be one of a list of possible values
 - could be a unique ID

Defining an Attribute

```
<!ATTLIST element_name  
  attribute_name attribute_type  
  default value>
```

Examples:

```
<!ATTLIST p color CDATA "black">
```

```
<!ATTLIST p size (small|large) "small">
```

Other attribute definition options

- You can also specify attributes as:
 - required (must be present)
 - implied (optional, no default value)
 - fixed (only one valid value)

Entities

- XML documents can include *entities*
 - variables used as shortcuts.
 - HTML has a fixed set of entities:
 - `&`; ` `; `<`;
- You can define entities within a DTD, and then use them in the XML document.
-

Declaring an entity

```
<!ENTITY entity_name entity_value>
```

Examples declarations:

```
<!ENTITY me "Dave">
```

```
<!ENTITY semester "Fall 2005">
```

Example usage in XML document:

```
<course>
```

```
<instructor> &me; </instructor>
```

```
<term> &semester; </term>
```

Back to XML

- It's perfectly fine to create XML documents that do not have a DTD
 - you don't have any way to formally validate the document.
- XML documents can include comments, they look just like HTML comments

```
<!-- anything you want here -->
```

XML and browsers

- You can view an XML file in a browser
 - needs to be a relatively modern version.
- Typically you will see hierarchical view of the document including tag names, attributes, etc.
 - in some browsers you can collapse levels in the hierarchy.
- Sample XML document: [Student Record](#)

XML and browsers

- A browser has no way of knowing how to draw an XML document (other than showing the hierarchy):
 - the tags don't mean anything to the browser, they are not like HTML tags which the browser understands.
- We can tell the browser how to display the tags in our XML documents.
 - CSS style rules

Some useful CSS properties for XML documents

- The CSS property `display` can be used to tell the browser to display an element as a block:
 - rendered within a bounding box
 - separated from surrounding content.
- Example Rule:

```
name { display:block; }
```
- You can use any CSS property (`display` just turns out to be useful for XML).

Student with CSS style

- Same XML file, but now includes this in the *prolog*:

```
<?xml-stylesheet type="text/css" href="stu.css"?>
```

- This is how you tell the browser about an external style sheet.

[student with CSS applied](#)

[The CSS style file \(viewable version\)](#)

CSS limitations

- Using CSS, you can't control the display of an XML document very much
 - can't change the order
 - can't restrict display of some elements
 - can't add additional content (headings, etc).
- There is a much more powerful form of stylesheet used with XML documents:
 - XSL Extensible Stylesheet Language

XSL

- An XSL stylesheet is itself an XML document.
 - There is a DTD that defines what an XSL document should look like.
- XSL allows you to add content, rearrange elements, duplicate elements, etc.
- XSL is actually more of a programming language than a style declaration.
 - There is a significant learning curve for XSL...

XSL Example

- The Same XML file (student record):

[Student with XSL style](#)

[XSL Style sheet \(viewable\)](#)

[All the sample files](#)

Exercises:

- Go through the tutorials at w3schools.com (as needed):
 - [XML](#)
 - [XML and CSS](#)
 - [DTD](#)
 - [XSL](#)
- Create an XML file that contains a list of the stolen music you own.
- Create a DTD, CSS stylesheet and XSL stylesheet.

Well Formed and *Valid* Documents

- A document that has proper XML syntax is called "Well Formed"
- A document that also has a DTD and obeys the rules in the DTD is called "valid"
- There are ,many stand-alone validation programs, as well as web-based validators.

XML Parsing

- Parse: to break down a document (or sentence) into the individual parts.
- If we want to extract information from an XML file we need to parse the file.
- There are some formal approaches:
 - DOM based
 - Event Based (SAX)

XML Parsing Libraries

- XML parsing has been standardized into the DOM and SAX approaches.
 - for almost any programming language you can find xml parsing libraries.
 - PHP supports both DOM and SAX parsing, we will look at both approaches.

DOM Parsing (PHP 5)

- The general idea is to give an XML parser an XML document (in memory, on a disk or sometimes even a URL)
 - The parser builds an OO representation of the XML document.
 - We can extract individual elements as long as we know where they are located in the XML document hierarchy (we usually do).
 - The DOM support relies on objects, which vary from one language to another...

DOM Example

```
<?xml version="1.0"?>
```

```
<student>
```

```
<name>
```

```
<first>Joe</first>
```

```
<last>Smith</last>
```

```
</name>
```

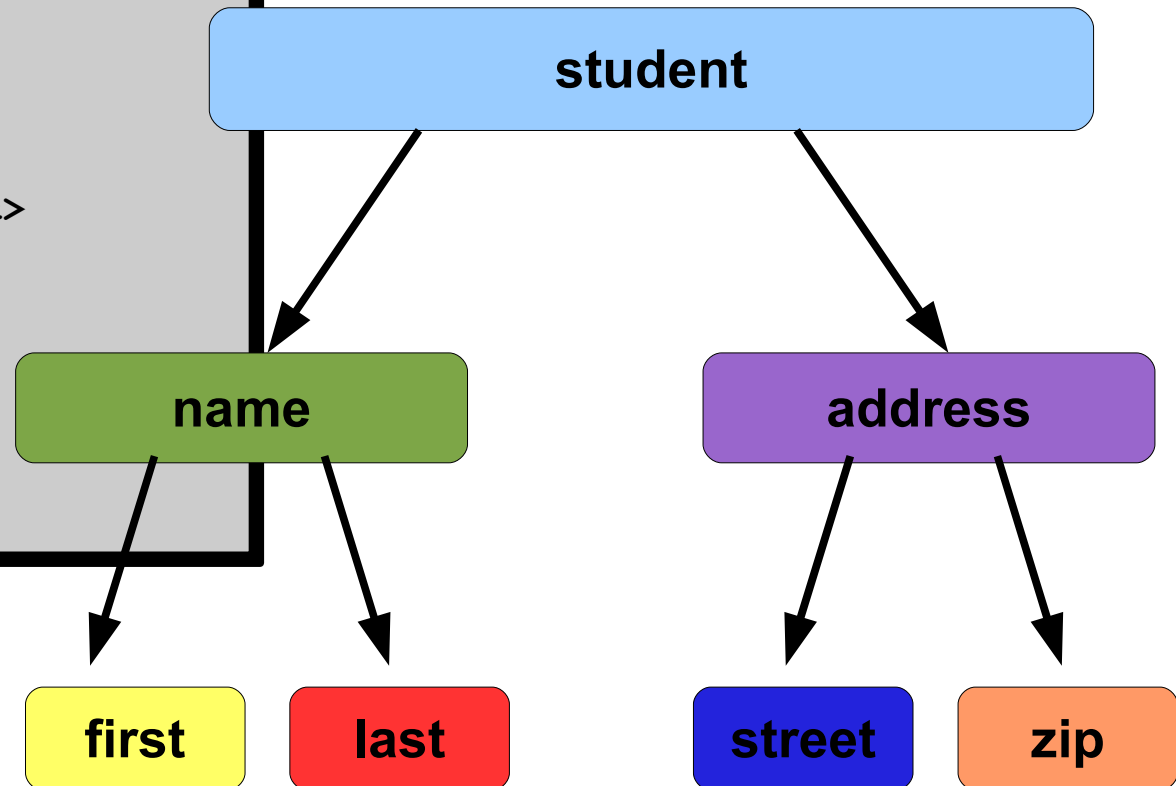
```
<address>
```

```
<street>110 8th st.</street>
```

```
<zip>12180</zip>
```

```
</address>
```

```
</student>
```



DOM Parsing and PHP

Uses PHP objects to represent XML Elements.

DOMDocument: represents an entire XML document.

DOMElement: represents an XML element

DOMNode: represents a node in the parse tree

a **DOMElement** is a **DOMNode**.

cdata is also represented by a **DOMNode**.

DOMAttribute: represents a tag attribute.

There are many more...

DOMDocument

- Constructor usage:

```
$doc = new DOMDocument ( ) ;
```

- Tell it to parse a file:

```
$doc->load( "somefile" ) ;
```

- Tell it to parse a string:

```
$doc->loadXML( "<foo> . . . </foo>" ) ;
```

Finding Elements

- You can ask a `DOMDocument` object to create a list of all the elements that match a tag name.
 - you can ask any `DOMElement` to do this as well...

```
$list=$doc->getElementsByTagName( "foo" );
```

`$list` is now a `DOMNodeList` object.

The length of the list is `$list->length`

item `$i` in the list is `$list->item($i)`

Getting at character data

- You can get all the cdata within a DOMNode (including any DOMELEMENT) with the nodeValue property.

```
$txt = $list->item($i)->nodeValue;
```

- Note that this is the CDATA for everything below the node (could be the CDATA from many tags).

Example

- Assume `$doc` represents the entire document, and `$n` represents the name element in the document.

```
<?xml version="1.0"?>
<student>
  <name>
    <first>Joe</first>
    <last>Smith</last>
  </name>
  <address>
    <street>110 8th st.</street>
    <zip>12180</zip>
  </address>
</student>
```

`$doc->nodeValue` is "Joe Smith 110 8th st. 12180"

`$n->nodeValue` is "Joe Smith"

Both would include a bunch of newlines...

PHP Dom Parsing Example

```
$doc = new DOMDocument();
$doc->load("example.xml");

$list = $doc->getElementsByTagName("name");

# the first name
$firstNode =
    $list->item(0)->getElementsByTagName("first")->item(0);
$firstname = $firstNode->nodeValue;

# the last name
$lastNode =
    $list->item(0)->getElementsByTagName("last")->item(0);
$lastname = $lastNode->nodeValue;

echo "Name: $firstname $lastname\n";
```

PHP DOM Parsing functions

- Other PHP DOM stuff:
 - not just access to elements/tags/attributes, also the ability to change them.
 - add new nodes/elements
 - generate XML from a DOM hierarchy.
- complete reference at
 - <http://www.php.net/manual/en/ref.dom.php>

SAX XML Parsing

- Respond to parsing event.
 - as the parser finds a specific tag it calls a function you write.
 - doesn't require memory to hold entire (document) tree at once.
- Parsing only, not useful for creating XML documents.

PHP Event Based XML Parsing

- Typical scenario:
 - Create a parser object
 - Tell the parser object what php functions to call whenever it parses a start tag, an end tag, or character data (three functions).
 - Open XML file for reading.
 - While stuff remains in file:
 - read next chunk of XML and give to parser.
- When parsing each chunk, the parser will call your functions when it finds a start tag, end tag or character data.

Creating a parser

```
$foo = xml_parser_create();
```

if you need namespace support:

```
$foo = xml_parser_create_ns();
```

foo is a PHP *resource*.

Registering tag *callback functions*

- To tell the parser to call functions named `my_start_handler` and `my_end_handler` when it finds start and end tags:

```
xml_set_element_handler($foo,  
                        'my_start_handler',  
                        'my_end_handler');
```

- Tells the parser what functions to call when it finds start or end tags.
- You need to write the handlers!

end tag handler function

- The end tag handler must have a prototype like this:

```
end_element_handler ( resource parser,  
                      string name)
```

When called, name is the tag name.

```
function my_start_handler($parser,$name) {  
    echo "Found end tag: $name\n";  
}
```

Example Start tag handler

```
function my_start_handler($parser,  
                           $name,  
                           $attrs) {  
    echo "Found start tag: $name\n";  
    echo "It has " . count($attrs) .  
    "attributes"\n;  
}
```

start tag handler function

- You write the callback functions that the parser will call.
- The start tag handler must have a prototype like this:

```
start_element_handler ( resource parser,  
                        string name,  
                        array attribs )
```

When called, name is the tag name and attribs is a list of attributes for that tag.

Register cdata callback

- Tells the parser to call the function `my_cdata_handler` whenever it finds character data.

```
xml_set_character_data_handler($foo,  
                              'my_cdata_handler');
```

cdata callback function

```
handler ( resource parser, string data )
```

An example:

```
function my_cdata_handler($parser,$cdata) {  
    echo "Found some data: $cdata\n";  
}
```

Sax Parsing Example

```
function startElement($parser, $name, $attrs) {  
    global $curtag;  
    $curtag = $name;  
}
```

```
function endElement($parser, $name) {  
    global $curtag;  
    $curtag = '';  
}
```

Sax Parsing Example (cont.)

```
function charData($parser,$cdata) {  
    global $curtag;  
    if ($curtag == "FIRST")  
        echo "Name is $cdata ";  
    else if ($curtag == "LAST")  
        echo "$cdata \n";  
}
```

NOTE: By default the parser will convert tag names to uppercase before calling any of the callback functions.

This is called case-folding (can be turned off).

Sax Parsing Example (cont.)

```
$xml_parser = xml_parser_create();  
xml_set_element_handler($xml_parser,  
    "startElement", "endElement");  
xml_set_character_data_handler($xml_parser,  
    "charData");  
  
// parser is now ready, it just needs a document.  
// We feed it a little bit at a time
```

Sax Parsing Example (cont.)

```
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}

while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die("XML error");
    }
}

xml_parser_free($xml_parser);
```

Examples

- Code samples including sample XML document and php parsing code (both DOM and Sax) are on the web

[Sample PHP Code](#)

Exercise

- Start with the XML file you created last class.
- Write a PHP program that parses the file and renders it as HTML.
 - create a table of songs (or whatever you created).
 - Add markup to make it look nice.
- In general you get the relevant CDATA and output it in HTML tags.
 - attributes can determine how something is displayed, etc.

