



XML - Beyond HTML

- XML eXtensible Markup Language allows the creation of markup languages for particular documents and particular domains
- XML tags describe the structure and semantics of a document's content, not the format of the content
 - the format of the content is described in a separate style sheet
 - XSL eXtensible Style Language specifically designed for use with XML
- DTDs Document Type Definitions are rules that describe how an XML document is structured
 - DTDs provide a list of the elements, tags, attributes and entities in the document, and their relationship to each other.



Hello XML!

Listing: `greetings.xml`

```
<?xml version="1.0" standalone="yes"?>
<greeting>
Hello XML!
</greeting>
```

- every XML doc begins with a declaration of version of XML in use. This is done in a *processing instruction* `<? ... ?>`
- processing instructions can also contain *attributes*: name-value pairs separated by `=` above `version="1.0" standalone="yes"`



XML Tags have *Semantic* and *Style* Meaning

- *Semantic* meaning is that which exists outside the document and is associated with the selection of the name of the tag `<greeting>`
- *Style* meaning specifies how the content of the tag is to be presented by the output device.
 - e.g. eXtensible Style Language, XSL



eXtensible Style Language, XSL

Listing: `greetings.xls`

<code><xls></code>	delimit the style sheet
<code><rule></code>	the first rule of the style sheet
<code><root></code>	“empty root tag” targets of this rule to the root elem. of XML
<code><H1></code>	
<code><children></code>	
<code></children></code>	everything contained in <code><H1></code> <code></H1></code>
<code></H1></code>	
<code></rule></code>	
<code></xls></code>	



XML's 7 forms of mark-up

1. Start and end tags `<rule>` `</rule>` Or `</rule>` empty tag
2. Attribute assignments name-value pair `version="1.0"`
3. Entity references
4. Comments `<!--` `-->`
5. Processing instructions e.g. include or exclude
6. Data section
7. Document type declaration `<!ELEMENT person (name, addr, tele)>`



XML Document Structure

1. The Prolog

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

2. The Document Type Declaration

```
<!DOCTYPE INVOICE SYSTEM "invoice.dtd">
```

3. The Document Type Definition

The DTD

4. The Root Element

the element that contains all others



DTD's Processing Instructions:

- DTD uses 4 kinds of markup declarations:
 - ELEMENT Declaration of an XML element type
 - ATTLIST Declaration of attributes that may be assigned to a specific element type *and* their permissible value(s)
 - ENTITY Declaration of re-useable content
 - NOTATION Format declaration for external content not meant to be parsed (binary data, for example) and the external application that handles the content



ELEMENT Type Declarations

- Provides a name for the declared type and a content specification, (names can begin with letter underscore (or colon))
- Content can be of 4 categories: **empty, element, mixed, any**
- **EMPTY** Empty element has neither text nor child element contained within it

```
<!ELEMENT SomeData EMPTY>  
instance of the type <SomeData/>
```

- **Element-only** element contains child but no text
- **Mixed** a mix of elements and parsed character data (**#PCDATA**) or text,content. Mixed and element content is indicated with a **content model** - the internal structure of an element's content.
- **ANY** lets the content of an element to be anything that does not violate XML well-formed syntax.

```
<!ELEMENT AnOldThing ANY> (use sparingly: parser can't do validity checking)
```



ELEMENT Type Declarations

- Allows an XML application to
 - constrain the elements that can occur
 - specify the order in which they can occur
- Sequence `<!ELEMENT person (name, addr, tele)>`
 - a `person` element consists of `name`, `addr`, `tele` in that order
- XOR Selection `<!ELEMENT animal (dog|cat|bear)>`
 - this is valid: `<animal> <dog> </animal>`
- zero or more `<!ELEMENT superstar (name*)>`
- Required (one or more) `<!ELEMENT failingstudents (name+)>`
- Any `<!ELEMENT person ANY>`
- Data `<!ELEMENT quotation #PCDATA>`
- You can also combine them!
 - `<!ELEMENT invoice (from, to (item+ | batch*))>`



ELEMENT Type Declarations, content model

- **Content model** consists of a set of parentheses enclosing some combination of child element names, operators (denote cardinality and indicate how elements and character data may be combined), and the **#PCDATA** keyword.

```
<!ELEMENT letters (A, (B | C))>
```

(letters has 2 child elements. 1st is always A, the 2nd is B or C)

- order operators

, (comma) strict sequence

| (pipe) choice (XOR)

- cardinality operators

? Optional,; may not appear

* Zero or more

+ One or more

```
<!ELEMENT FruitBasket (Cherry+, (Apple | Orange)*)>
```



ELEMENT Type Declarations, content model

```
<!ELEMENT FruitBasket (Cherry+, (Apple | Orange)* )>
```

A conforming instance of **FruitBasket**

```
< FruitBasket>  
  <Cherry> ... </Cherry>  
  <Cherry> ... </Cherry>  
  <Apple> ... </Apple>  
  <Orange> ... </Orange>  
  <Orange> ... </Orange>  
< /FruitBasket>
```



Attributes

Attributes complement and modify elements by providing a means of associating simple properties with the element

ATTLIST declaration consists of; **ATTLIST** keyword , element name, zero or more attribute definitions.

```
<!ATTLIST myElement attributeName CDATA #REQUIRED >
```

Defaults for attribute declaration

#REQUIRED The attribute must appear in every instance of the element

#IMPLIED The attribute may optionally appear in every instance of the element

#Fixed (plus default value) The attribute must always have default value , if it does not appear the value is assumed by the parser



ENTITY Type Declarations

- Facility for declaring chunks of content and referencing them as many times as you want
- Defines the **name** and **content** You refer to it by **name**
- Pre defined Entities:

- Special Characters

<	<
>	>
&	&
'	'
“	"



ENTITY Type Declarations, Cont.

- General Entities

declaration: `<!ENTITY copyright "Princeton Univ, 2000">`

reference: `©right;` (parser displays Princeton Univ, 2000)

- Can also have an external form:

declaration: `<!ENTITY myEntity SYSTEM "http://www... >`

(Keyword **SYSTEM** followed by URL)

reference: `&myEntity;`

- Entities can not refer back to themselves



ENTITY Type Declarations, Cont.

- Parameter Entities

Parsed Entities used solely within DTD. They let us easily reference and/or change commonly used constructs in the DTD by keeping them in one place.

declaration: `<!ENTITY % vehicleParameters "driver CDATA #IMPLIED location CDATA #REQUIRED speed CDATA #REQUIRED">`

(declaration must be made *before* they are referred to in the DTD)

reference: `<!ATTLIST tripInfo %vehicleParameters; destination CDATA #REQUIRED>`

Same as if they were all listed out



Example: XML

- NaVigation Markup Language
 - A markup language for describing navigation information
 - Based on XML



NVML (cont.)

- Capabilities:
 - Route Navigation
 - Points of Interest
- Elements
 - `<head />`
 - `<body />`



NVML (cont.)

- Children of <head/>
 - title
 - category
 - transport
 - duration
 - distance
 - expense
 - geodetic-system
 - info



NVML (cont.)

- Children of <body/>
 - navi, guide
- Children of <navi />
 - point, route, info
- Children of <guide />
 - point, info



NVML Example

```
<navi>
  <route>
    <name>Bell Highway</name>
    <category>highway</category>
    <means>car</means>
    <duration>10 minutes</duration>
    <distance>12 mi</distance>
    <expense>50 cents</expense>
  </route>
  <info duration="20">
    <text> Bell Highway </text>
    <voice> This is Bell Highway. </voice>
    <image src="image/bell-highway.gif"/>
  </info>
</navi>
```



NVML Example (cont.)

- [Text Only Display](#)
- [In-Vehicle Computer](#)
- [Handheld Computer](#)
- [PCMDemo](#) The WSDL - <http://pcmws.alk.com/?wsdl> alksales *****password alksalesacc
- [CoPilot|Live](#)



Applet: Web page embedded Java program

- Applet runs locally (security is paramount)
 - runs on client machine running browser, NOT on system running HTTP server
- Applets are restricted from performing many normal JAVA ops
 - Don't read from client disk
 - can instruct browser to display pages that are generally accessible on the Web which might include some local files (via cookies)
 - Don't write to local disk
 - browser can still cache certain files including those loaded by the applet (but they are not under the direct control of the applet)
 - Don't open network connections (other than to the Server from which the applet is loaded; prevents browsing behind firewalls)
 - Do NOT call local Programs
 - Can NOT discover private info (user name), but can determine name of host because this is already reported to the HTTP server that delivered the applet



Creating an Applet

1. Make the Java class

Defines the actual behavior of the applet

2. Make the associated HTML document

Associates the applet with a particular rectangular region of the Web page



Java Applet template

1. Section for the declaration of instance variables

2. An **init** method

Automatically called by browser when applet is first created

3. A **paint** method

called after **init** has finished and

whenever image has been obscured

when graphical components are added

programmatically



Applet life cycle

- Has no **main** method
- **public void init()**
 - only called after applet instance is first created by browser
 - not called again
- **public void start()**
 - called by browser
 - after **init()** is completed
 - after applet has been stopped
 - called to restart animation
- **public void paint(Graphic g)**
 - user level drawing called after start is completed and when browser wants to repaint (obscured window)
 - **repaint()** user call to restart **paint(Graphic g)**



Applet life cycle (con't)

- **public void stop()**
 - browser call when user leaves the page or minimizes browser. Can be used to halt animation
- **public void destroy()**
 - called by browser to keep images from getting too large and to free resources
- host of other methods for graphics and other things



HTML Template

Required attributes: **CODE**, **WIDTH**, **HEIGHT**

CODEBASE (defaults to directory containing class file) otherwise, is path to class file directory

Java source file need not be available on Web.

All non-system class files used by applet need to be WWW accessible and should be in either the same directory or subdirectory to class file. (if you use two custom classes they must be moved with the applet)

System classes are ALWAYS loaded from the client

Applet must be declared public



HTML Applet TAG

```
<APPLET CODE="..." WIDTH="..." HIGHT="..." ...>  
... </APPLET>
```

CODEBASE, ALT, ALIGN, HSPACE, VSPACE, NAME

HTML Element: `<PARAM NAME="..." VALUE="...">` (NO
END TAG)

In *.JAVA: `getParameter("ALAIN")`

In *.HTML: `<PARAM NAME="ALAIN" VALUE="MAYBE">`