WTP-101

Developing Web Applications with Standards

using W3C org standard technologies such as, HTML, CSS, XML, XSD and XSL
Attributions

• **World Wide Web Consortium**
  – http://www.w3c.org

• **Sandra Clark**
  – CSS for Better Sites – CFUN04
  – http://www.cfconf.org/
# Web Standards

## Module Road Map

<table>
<thead>
<tr>
<th>Web Standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Architecture: Resources, URI and HTTP</td>
</tr>
<tr>
<td>HTML and XHTML</td>
</tr>
<tr>
<td>XML, XML Schemas and XML Parsing</td>
</tr>
<tr>
<td>CSS</td>
</tr>
<tr>
<td>XSLT</td>
</tr>
</tbody>
</table>
What are Web Standards

• **Worldwide Web Consortium (W3C)**
  – Recommends Standards for Web Development

• **Recommendations:**

  *http://www.w3.org*

Specifications for the Web's formats and protocols must be compatible with one another and allow (any) hardware and software used to access the Web to work together.
w3c.org – The “one” web

• The W3C Technology Stack

Figure: http://www.w3.org/Consortium/technology
What Standards?

• Standards for the Web means:
  – Structural Languages
    • HTML – Publishing Language of the Web
    • XHTML - Extensible Hypertext Markup Language 1.0 and 1.1
    • XML - Extensible Markup Language 1.0
  – Transformations
    • XSL - Extensible Stylesheet Language
    • XPath – XML Path Language
  – Presentation
    • CSS - Cascading Style Sheets Levels 1 and 2
  – as well as emerging standards, such as those for television and PDA based User Agents
Web standards are important

• Designing and building with Web standards
  – Simplicity
    • Simplifies and lowers the cost of production
  – Accessibility
    • Delivers sites that are accessible to more people
    • Delivers sites that are accessible more types of Internet devices.
  – Continuity
    • Sites will continue to function correctly as traditional desktop browsers evolve, and as new Internet devices come to market

Quoted from http://www.webstandards.org mission statement
XML, HTML, XHTML, XSL & CSS

• **XML for content**
  - Most portable way to share and transfer information

• **HTML/XHTML for publishable document structure**
  - Structure does matter

• **XSL for transformation**
  - Transform between document types

• **CSS for presentation**
  - If it isn’t content it doesn’t belong in HTML
Standards Related

Figure: http://www.w3c.org
Web Architecture

Module Road Map

- Web Standards

- **Web Architecture: Resources, URI and HTTP**
  - HTML and XHTML
  - XML, XML Schemas and XML Parsing
  - CSS
  - XSLT
Section Goals

• To learn basic Web architecture
• To learn how Resources, URI and HTTP are used to access information on web servers
Simple Web Architecture

URI

http://www.eclipse.org/webtools/education/101

Representation

Metadata:
Content-type: application/xhtml+xml

Data:
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>Web Tools Platform</title>
</head>
<body>

</body>
</html>

WTP 101
Web Application Development

Resource
Http: Protocol of the Web

• The Internet consists of servers, clients, and routers
  – Servers provide the information
  – Clients use the information on the servers
  – Routers provide the network that allows clients and servers to communicate

• Clients and servers typically communicate using HyperText Transfer Protocol (HTTP)
Simple HTTP

- **URI:**
  - The browser connects to the Web Server using a socket
  - The browser sends a “GET” request

- **Resource:**
  - The server resolves the request
  - Standard web pages are produced by the server

- **Representation:**
  - The HTML is sent to the browser
  - The socket is closed; the browser renders the document using HTML
Atomic Requests

- HTTP requests are non-conversational
  - A different socket is used to satisfy each request
- Traditional HTTP provides no mechanisms for multiple request relationships with clients
  - Cookies can be used to maintain information about the client’s identity
What is an URL?

- Uniform Resource Locator, or an address pointing to an Internet resource

```
http://www.eteration.com:80/page.html
```

- Scheme (http, ftp, gopher, ...)
- Port
- Name of server
- Resource (name of page to download)

If you don’t specify a port, 80 is assumed.
URLs

• A URL specifies the identity of the computer as well as the required resource
• File resources are specified relative to a “web root”
  – The “web root” is a directory on the server
  – The resource may include subdirectory information

http://localhost:7001/stuff/page.html

Access the file /stuff/page.html from the server running on port 7001 on the local computer

The URL may be shown at the top of your browser as the "location" or "address"
Clients

• Clients access information provided by the servers
  – Web browsers are probably the most common web clients

• A client requests files by sending a HTTP request to server
  – The request is sent over the internet using sockets
  – The file is specified in the request using a Uniform Resource Locator (URL)
HTTP Request

- The request is formed by the client to inform the server of the request
- The request header includes:
  - Supported HTTP version, type of the requestor (User-agent), accepted formats (Accept), accepted languages, cookies, ...

```
http://localhost:8080/stuff/page.html
```

```
Get /stuff/page.html HTTP/1.1
Accept: text/html
Accept-Language: en-us
User-Agent: Mozilla/4.0
```

sent to “localhost:8080”
Servers

• Web servers provide information to web clients
  – When a request comes in from a client, the server “serves” a response

• The response contains header information as well as the content of the page

• The type is contained in the header
  – This specifies what type of information is being returned in the response (HTML page, an image, sound file, ...)
  – The client uses the type to decode the information in the response and present it to the user
HTTP Response

• The server’s response includes a header followed by content data
  – The client uses information in the header to determine what to do with the content

• The response header includes
  – Content type, content length, cookies, ...

Server: JavaWebServer/1.0
Content-Type: text/html
Set-Cookie: id=954096

<HTML>
<BODY>
Hello World!
</BODY>
</HTML>
What is MIME?

• **Multipurpose Internet Mail Extensions protocol**
  – Standard for identifying and encoding binary data for transmission
  – Originally designed for sending e-mail attachments

• **HTTP uses MIME**
  – Identify the type of object in the response
  – Typically “text/html” which indicates that the return value is an HTML document

• **Browsers use this information to decide what to do with the content**
  – MIME also specifies a number of different encoding schemes for transporting 8-bit data over 7-bit protocols

**MIME Encoding is not part of this course.**
Some MIME Types

- Content types are specified as a type/subtype pair
  - Both the type and subtype are required
- **text/html**
  - The content of the message is HTML-formatted text
- **text/plain**
  - The content of the message is unformatted text
- **image/jpeg**
  - The content of the message is a JPEG image

MIME Types are case insensitive.
Cookies

- **Servers return additional information to the client via cookies**
  - Clients return the cookie information on subsequent requests
- **Cookies can be used to maintain a relationship between a browser and the server**
- **Cookie’s life span can be configured**
  - Live until a specified date and time
  - Live until the browser closes
Web Pages

- Web pages consist of text and HTML tags which provide formatting “suggestions” to web browsers
- Pages may contain images, movies, sounds, and other types of multimedia
- Pages may also contain client-side technologies
  - Java applets, JavaScript, ActiveX components which are downloaded and executed on the client
- Pages can provide links to other pages
  - Links allow a user to move quickly and easily between related web pages

A web site is a collection of related web pages.
Pages Can Be Static or Dynamic

• A web page may be an actual file located on a server
  – Static content

• Web pages may also be dynamically generated by the server
  – Java servlets, Java Server Pages (JSPs)
  – Many, many others!
Dynamic Content

- Servlets and JSPs are accessed using a request with a URL - just like a regular page.
- Unlike a regular page, the content in the response is generated dynamically by the servlet or JSP.
- Servlets don’t just generate HTML!
  - Servlets can also be used to generate other MIME Types such as images.
- Servlets are the subject of another course!
What You Have Learned

• The Internet consists of servers, clients, and routers
• Web clients access information on web servers using HyperText Transfer Protocol (HTTP)
• Web pages contain text and multimedia
• Web pages may be static or generated dynamically
Hands-On Lab

- Setup a Preview Server
  - Software is provided with WTP

- Create a simple page
  - Hello world will suffice

- Monitor HTTP traffic with TCP-IP Monitor
  - TCPIP Monitor is a proxy between the browser and the server
HTML and XHTML

Module Road Map

- Web Standards
- Web Architecture: Resources, URI and HTTP
- HTML and XHTML
- XML, XML Schemas and XML Parsing
- CSS
- XSLT
Section Goals

• To learn Web standards for HTML and XHTML
• To learn the structure of an HTML document
• To learn how to use basic HTML tags
HTML Overview

- HTML stands for HyperText Markup Language
- HTML files consist of text and tags
  - Text provides the content of the page
  - Tags provide formatting "suggestions" to the client
    - It is up to the client how these suggestions are implemented
- HTML tags are case-insensitive
- Whitespaces within HTML files are generally ignored
  - Formatting tags are used instead to specify line breaks, indentation, etc.
XHTML

• XHTML is an xml compliant version of HTML 4.01
• Benefits of using XHTML
  – Easier to validate against
  – Because its more stringent, we are more careful
  – Requires the use of CSS for all presentation.
  – Standard across most User Agents
HTML vs. XHTML

- **Element and Attributes**
  - HTML
    - `<H1></H1>`
  - XHTML must be lowercase
    - `<h1></h1>`
  - `<input type="Hidden">`
  - `<input type="hidden" />

- **End tags are required**
  - HTML
    - `<p>`
  - XHTML
    - `<p></p>`

- **Empty Elements**
  - HTML
    - `<br>`, `<hr>`
  - XHTML
    - `<br/>`, `<hr />`

- **Quotes**
  - HTML
    - `<input type=Hidden value='myvalue'>`
  - XHTML
    - `<input type=“hidden” value=“myvalue” />`

- **name/value pairs**
  - HTML
    - `<input type="checkbox" checked>`
  - XHTML
    - `<input type=“checkbox” checked=“checked”/>`
DOCTYPE

• XHTML Documents must be well formed
  – MUST start with a <!DOCTYPE>

• User Agents (browsers) use the DOCTYPE
  – Choose what mode to use when rendering your HTML

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Web Tools Platform</title>
  </head>
  <body>
    ...
  </body>
</html>
```
Which mode am I in?

- To check which Rendering mode your computer is in, use the following:
  - IE6 – Opera
    - `javascript:alert(document.compatMode);`
    - CSS1CompatMode – Standards Based Rendering
  - Firefox, Mozilla – Netscape
    - CTRL-I for page information.
Forcing User Agents

• Force Standards Mode
  – Example: HTML 4.x Strict
    ```html
    <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
    ```

• Quirks Mode - XML declaration with the DocType
  – You need to use features from browser supports
    • will Force IE6 and Opera into Quirks Mode
  – Avoid using `<?xml version="1.0" encoding="UTF-8"?>`
    • Stay in standards mode
HTML Tags

• Most tags have a start tag that indicates the start of the formatting and an end tag to specify the end
  – Start tags are of the form `<tag>`
  – End tags are of the form `</tag>`

• The formatting applies to the text between the start and end tag

• Some tags also have attributes which provide more information within the start tag
  – Attribute values may use single or double quotes
  – Single quotes will make your life easier later...

This shows some `<b>bold</b>` text.

This shows some **bold** text.
Page Structure Tags...

• Tags used to specify the structure of the page
  – Pages have a head and a body

• Pages start with a `<html>` and end with a `</html>`
  – Tells the browser what type of file it is

• The `<head>` tag comes at the top of the page
  – May contain a `<title>` tag
    causes the window name to be changed while the page is being displayed

• The `<body>` tag follows the `<head>` tag
  – The body contains the content of the page
...Page Structure Tags

<html>
  <head>
    <title>Page title</title>
  </head>
  <body>
    <h1>Header</h1>
    ...Page content...
    <h2>Subtitle</h2>
    ...More content...
  </body>
</html>
Basic Formatting Tags...

- <!-- ... --> - Comment
- <b> - Bold text
- <i> - Italicized text
- <u> - Underlined text
- <br/> - Add a line break to the text
- <hr/> - Add a line break and header rule
- <p> - Paragraph
  - <p align="right"> - Start a new right-justified paragraph
- <h1> - Text is formatted as a level-1 heading
  - Can also use <h2>, <h3>, <h4>, <h5>, and <h6>
- <center> - Text contained in these tags is center-justified

By default, text is left-justified
...Basic Formatting Tags

<html>
<head>
<title>Eteration!</title>
</head>
<body>
<h1>Welcome to Eteration!</h1>
<hr />
<p>Training<br />
Consulting</p>
<p>Products</p>
<hr />
</body>
</html>

Welcome to Eteration!

Training
Consulting
Products

Paragraph tags should have an end tag!
Table Tags

- A table is specified by providing tags for each row; the columns are specified with each row

Tags:
- `<table>` - Creates an HTML table
- `<tr>` - Starts a new row within a table
- `<td>` - Starts a new cell within a table row
- `<th>` - A heading cell within a table

```html
<table border="2">
  <tr><th>Employee</th><th>ID</th><th>Phone #</th></tr>
  <tr><td>Tom Johnson</td><td>45938</td><td>432-7548</td></tr>
  <tr><td>Steve Smith</td><td>12450</td><td>349-9832</td></tr>
  <tr><td>Dan Jones</td><td>34545</td><td>887-3492</td></tr>
</table>
```
HTML Lists

- **HTML has tags that output text in a list format**
  - `<ul>` - Unordered (bullet) list
  - `<ol>` - Ordered (numbered) list
  - `<li>` - Start a new entry in a list (ordered or unordered)

Shopping list:
```html
<ul>
  <li>Oranges</li>
  <li>Bananas</li>
  <li>Faux-fu (Tofu substitute)</li>
</ul>
```

Things to do:
```html
<ol>
  <li>Do groceries</li>
  <li>Get a hair cut</li>
  <li>Clean the house</li>
</ol>
```

- Shopping list:
  - Oranges
  - Bananas
  - Faux-fu (Tofu substitute)

- Things to do:
  1. Do groceries
  2. Get a hair cut
  3. Clean the house
HTML Links

• Create a hyperlink using the `<a>` tag
• This tag has one attribute call `href`
  – Used to specify the URL of the location to link to
• The link can refer to an HTML page, a servlet, an image, ...

Click
`<a href="http://www.eteration.com/education/">here</a>`
to go to education pages.

White space is ignored by HTML formatters
The Image Tag

- Image tags are used to display graphical images
- The image tag can have a number of different parameters
  - "src"
    - The source URL of the image; the browser will use this URL to make a request for the image
  - "alt"
    - Specifies alternative text to display if the browser can’t (or won’t) display the graphic
  - "height" and "width"
    - Used to customize the size of the image without altering the source file

```html
<img src="images/iteration400.gif" alt="Eteration Logo"/>
```
What You Have Learned

- How a web page is structured
- How to use basic HTML tags
- How to add lists and hyperlinks to your HTML pages
Hands-On Lab

• Create a Web page
  – XHTML Transitional 1.0
  – Validate XHTML at http://validator.w3.org/

• Use tables for layout

• Use tables for listing objects

• Tables are very complex to work with.
  – We will fix some of the problems later
XML and XML Schemas

Module Road Map

- Web Standards
- Web Architecture: Resources, URI and HTTP
- HTML and XHTML
- XML and XML Schemas
  - CSS
  - XSLT
Section Goals

• To learn about XML
• Compare HTML, SGML and XML
• To learn about DTDs
• To learn about XSDs
• To learn basic XML parsing techniques and APIs
Common Terms

• XML: eXtensible Markup Language
• XSD: XML Schema Definition
• DTD: Document Type Definition
SGML Background

• Standard Generalized Markup Language (ISO 8879)
  – Motivated by heavy document processing requirements of large organizations
  – Exchange text without losing “structure”
  – Complex failed to gain wide acceptance

• Both XML and HTML came form SGML
Format Markup vs. Structure Markup

- **Meaning comes with structure**
  - How can you tell the name of this person?

```
Naci Dai
eteration a.s.
25 ITU ARI-1 Teknokent
Maslak Istanbul
34469
Turkey
```

**Format**
- Arial 24pt Bold
- Lucida 24pt Orange
- Times Roman 18pt
- Times Roman Bold 18pt
- Times Roman Bold 18pt

**Structure**
- Name
- Company
- Address
- Postal Code
- Country

**Markup** Identifies Elements of a Document
HTML is Limited

• Simple markup language
  – Not designed for structuring data
• Result:
  – Not for arbitrary universal custom data

Web

Evolution

Integrate
Remote heterogenous Systems

Structure
Arbitrary Data

Multi Channel Delivery
different Presentation media
What is XML?

- **E**xtensible **M**arkup **L**anguage
- XML is a *metadata* language

**Data is:**
- Web page
- Printed Book
- Product

**Metadata is:**
- Information about data (data about data)
- Describing what the data is, identifying content
XML is Extensible

- Define your own tags
  - There is no single set of XML tags
  - Unlike HTML, where there is a core set of tags
- Comprimising extensibility HTML is easy to learn and use

```
<html>
  <head>
    <title>Eteration</title>
  </head>
  <body>
    <h1>Hello!</h1>
    <address>
      <name>Naci Dai</name>
      <company>Eteration</company>
      <street>25 ARI-1 ITU Teknokent</street>
      <zip>34469</zip>
      <country>Turkey</country>
    </address>
    <message>
      <text>Hello</text>
    </message>
    <order>
      <price>10$</price>
    </order>
  </body>
</html>
```
XML is for Markup

• Markup is identifying distinct elements of documents
  – Essential for documents to make sense

```
<?xml version="1.0" encoding="UTF-8"?>
<Address>
  <Name>Naci Dai</Name>
  <Company>Eteration</Company>
  <Suite>25</Suite>
  <Street>ITU ARI-1 Teknokent</Street>
  <Zip>34469</Zip>
  <City>Istanbul</City>
  <Country>Turkey</Country>
</Address>
```
XML is a Language

- XML is a formal document markup language
- A document has a physical and logical structure
  - Physical:
    - Composed of units called entities that may refer to others
    - There is a "root" or document entity
  - Logical
    - Composed of declarations, elements, attributes, comments, character references, and processing instructions
- XML has syntax
  - Indicated in the document by explicit markup
  - The logical and physical structures must nest properly
XML Elements

Element describes data
- One can define any element
- Element can contain other elements
- An element is terminated by </…>

<invoice>
  <from>ABC TELECOM, Inc.</from>
  <to>John Smith</to>
  <description>Local Phone Service</description>
  <date type="from">16 May 1999</date>
  <date type="to">15 Jun 1999</date>
  <date type="due">15 Jul 1999</date>
  <amount>$50.00</amount>
  <taxRate>6</taxRate>
  <totalDue>$53.00</totalDue>
</invoice>
XML Attribute

• **Describes an element**
  – One can define any attribute
  – Cannot contain other elements or attributes

```xml
<?xml version="1.0" encoding="UTF-8"?>
<invoice type="bill" period="monhly">
  <from>ABC TELECOM, Inc.</from>
  <to>John Smith</to>
  <description>Local Phone Service</description>
  <date type="from">16 May 1999</date>
  <date type="to">15 Jun. 1999</date>
  <date type="due">15 Jul. 1999</date>
  <amount currency="USD">$50.00</amount>
  <taxRate>6</taxRate>
  <totalDue>$53.00</totalDue>
</invoice>
```
Grammars for XML Documents

• Two current standards for constraining XML with grammars
  – DTD (Document Type Definition)
  – XML Schema
DTD: Document Type Definition

• **DTD**
  - defines document structure
  - makes XML data usable for different programs
  - can be declared inline or as external reference

• **Internal DOCTYPE declaration**
  - `<!DOCTYPE root-element [element-declarations]>`

• **External DOCTYPE declaration**
  - `<!DOCTYPE root-element SYSTEM "filename">`

**Hint:** When possible use XML Schemas
DTD Example

<?xml version="1.0"?>
<!DOCTYPE email [ 
<!ELEMENT email (to+, from, subject, message)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT subject (#PCDATA)>  
<!ELEMENT message (#PCDATA)> 
]>  

email.dtd

<?xml version="1.0"?>
<!DOCTYPE email SYSTEM "email.dtd">
<email>
  <to>info@eteration.com</to>
  <from>webmaster@eteration.com</from>
  <subject>Important</subject>
  <message>Hello!!!</message>
</email>

email.xml
XSD : XML Schema Definition

• XSD : XML Schema Definition
  – Is an XML language for describing and constraining the content of XML documents.
  – Alternative to DTD

• XSD: specifies structure of XML document i.e.
  – elements and attributes in the XML doc
  – XML element hierarchy
  – element data-types and occurrences

• http://www.w3.org/2001/XMLSchema
Types and Elements

• **XSD schemas contain type definitions and elements**
  – Type definitions define XML data type
    • address, customer, purchaseOrder,...
  – Elements represent items created in the XML file
    • If the XML file contains a PurchaseOrder type, then the XSD file will contain the corresponding element named PurchaseOrder.
XSD template

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.etereation.com"
    xmlns:tns="http://www.etereation.com"
    elementFormDefault="qualified">
</xs:schema>

1 – Elements and data-types used come from here. Prefix these elements with xs
2 – Elements defined in this schema have this namespace.
3 – Default namespace
4 – Must be namespace qualified
XML referencing an XSD

- Corresponding xml references xsd.
- Validation checks formation and cross checks XML against XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<m:message
    xmlns:m="http://www.example.org/message"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.example.org/message message.xsd">
    <m:to>Derya</m:to>
    <m:from>Esma</m:from>
    <m:subject>Please call</m:subject>
    <m:text>Call me ASAP</m:text>
</m:message>
```

xsd namespace
ref xsd file
Namespaces

• **XML Namespaces provide a method to avoid element name conflicts**
  – a name conflict will occur when two different documents use the same element names.

• **Every XML Schema uses at least two namespaces**
  – targetNamespace
  – XMLSchema namespace
    • http://www.w3.org/2001/XMLSchema
Need for Namespaces

`<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<book>
    <!-- title of a book -->
    <title> Eclipse Web Tools Platform</title>
</book>

<!-- title of a figure -->
<figure>
    <title>Simple Web Architecture</title>
</figure>`

Ambiguous

With namespace
Namespace Syntax

• Two parts
  – Namespace declaration
  – Elements and attributes

• Declaration
  – A prefix is associated with URI
  – The association is defined as an attribute within an element
    - xmlns:prefix
  – xmlns is Namespace keyword, prefix is user-defined

```xml
<classes xmlns:XMLclass="http://www.example.org/test">
  <XMLclass:syllabus>
    ...
  </XMLclass:syllabus>
</classes>
```
Namespace Declaration

• **Can be declared in:**
  – root element
  – lower level element

• **Multiple different namespaces can be defined**

• **Same prefix can be redefined**
  – Scope of Namespace declaration is within the element where it is defined
Elements and attributes

• Examples
  – svg:set
  – mathml:set

• **prefix: local part**
  – prefix identifies the namespace an element and attribute belongs to
  – local part identifies the particular element or attribute within the namespace
  – Qualified name

• **Naming rules:**
  – **Prefix** can be composed from any legal XML name character except “:”
  – “xml” (in any case combination) is reserved so cannot be used as prefix
  – **Local** part cannot contain “.”
Namespace URI

• **URI cannot be prefix**
  - “/”, “%”, and “~” are not legal in XML element names

• **URI could be standardized**
  - (by industry standard orgs) while prefixes are just convention

• **URI are just “identifiers”**
  - URI does not have to be in “http” form
  - URI does not have to be resolved
  - It is like a “constant value”
Default Namespace

- **Denoted with `xmlns` attribute with no prefix**
  - Applied only to unprefixed element and its descendant elements
- **Applies only to elements not attributes**

```xml
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
     xmlns:xlink="http://www.w3.org/1999/xlink">
  <head>
    <title>Three Namespaces</title>
  </head>
  <body>
    <h1 align="center">An Ellipse and a Rectangle</h1>
    <svg xmlns="http://www.w3.org/2000/svg"
         width="12cm" height="10cm">
      <ellipse rx="110" ry="130" />
      <rect x="4cm" y="1cm" width="3cm" height="6cm" />
    </svg>
  </body>
</html>
```
Types of Namespaces

- **target Namespace**
  - Namespace for XML Schema document itself

- **source Namespaces**
  - Definitions and declarations in a schema can refer to names that may belong to other namespaces

```xml
<xsd:schema
    targetNamespace='http://www.SampleStore.com/Account'
    xmlns:xsd='http://www.w3.org/1999/XMLSchema'
    xmlns:ACC= 'http://www.SampleStore.com/Account'>

<xsd:element
    name='InvoiceNo'
    type='xsd:positive-integer'/>

<xsd:element
    name='ProductID'
    type='ACC:ProductCode'/>

<xsd:simpleType
    name='ProductCode'
    base='xsd:string'>
    <xsd:pattern value='[A-Z]{1}d{6}'/>
</xsd:simpleType>
</xsd:schema>
```
The namespace that is assigned to the schema created
– The names defined in a schema are said to belong to its target namespace
– The namespace an instance is going to use to access the types it declares

Each schema has:
– One target namespace
– Possibly many source namespaces
Defining Types

- **Types may be simple or complex**
  - **SimpleTypes**
    - cannot contain elements or have attributes
    - are types that are included in the XML Schema definition (boolean, string, date, etc.)
  - **ComplexType**
    - can contain attributes and elements
Common XML Schema Data Types

- string
- boolean
- decimal
- float
- double
- duration
- dateTime
- time
- date
XSD: SimpleType Example

- Describes the data allowed in a Simple Field:

  <simpleType name="name">
    <restriction base="string">
      <xs:pattern value="([a-z][A-Z])+"/>
    </restriction>
  </simpleType>

- More Restriction Specs:

  <xs:restriction base="xs:integer">
    <xs:minInclusive value="0"/>
    <xs:maxInclusive value="100"/>
    <xs:pattern value="[0-9][0-9][0-9]"/>
  </xs:restriction>

- Constraints: enumeration, length, minLength, whitespace etc.
XSD: ComplexType Example

• **Similar to defining a Java class or a Data Structure**
  – Can use own types

```
<complexType name="PersonType">
  <sequence>
    <element name="name" type="string"/>
    <element name="surname" type="string"/>
    <element name="address" type="tns:AddressType"/>
    <element name="phoneNumber" type="tns:PhoneType"/>
  </sequence>
</complexType>
```
Type & Element

• Name the Type if it will be used again

```xml
<xs:complexType name="AddressType">
  <xs:sequence>
    <xs:element name="street1" type="xs:string" />
    <xs:element name="street2" type="xs:string" />
    <xs:element name="postcode" type="xs:string" />
    <xs:element name="city" type="xs:string" />
  </xs:sequence>
</xs:complexType>

<xs:element name="shippingAddress" type="tns:AddressType" />
<xs:element name="invoiceAddress" type="tns:AddressType" />
```
XSD: Indicators

• **Order**
  – all
    • Not ordered
  – choice
    • One of
  – sequence
    • Ordered

• **Multiplicity**
  – `minOccurs` / `maxOccurs` - Use unbounded for open boundary

```xml
<xs:choice>
  <xs:element name="employeeName" type="xs:string"/>
  <xs:element name="employeeNum" type="xs:integer"/>
</xs:choice>
```

```xml
<xs:element name="person" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string" minOccurs="0" maxOccurs="5"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```
Importing a Schema

• **Reuse and refactor XSD documents**
  – Partition namespaces
  – Use existing schemas

• **Import**
  – XSD is not same namespace

• **Include**
  – XSD is the same namespace

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:store="http://www.store.com/store"
    xmlns:catalog="http://www.partner.com/catalog">

  <xs:import namespace='http://www.partner.com/catalog'
     schemaLocation='http://www.partner.com/catalog.xsd'/>

  <xs:element name='stickyGlue' type='catalog:SuperGlueType'/>
</xs:schema>
```
What You Have Learned

• **XML**
  – standard for data interchange
  – was designed to describe data and to focus on what data is
  – text-based
  – does not define tags of its own
Hands-On Lab

- Create a Schema for the ObjectShop catalog
  - Use WTP XSD Editor
- Create Sample Catalogs
- Validate catalog files using XSD
CSS

Module Road Map

- Web Standards
- Web Architecture: Resources, URI and HTTP
- HTML and XHTML
- XML and XML Schemas
- CSS
- XSLT
What is CSS?

• **CSS:** Cascading **Style Sheets**
  – The method used to divide the content from the presentation on web pages.

• **Styles**
  – define **how to display** HTML elements
  – normally stored in **Style Sheets**
Recall: Standards Related

Figure: http://www.w3c.org
CSS Design Benefits

• **Maintenance and Flexibility**
  - Cleaner / Less code
  - Refactor presentation reduce repetitive styling
  - Better document structure

• **Accessible**
  - Structure is separated from presentation
  - Ability to present content on multiple devices such as mobile handhelds and formats (printer-friendly etc.)

• **Faster download times and smaller pages**
  - Tableless layouts, no repetition, all styles in one place
CSS Syntax

- The CSS syntax is made up of two parts:
  - Pattern
  - Rule

- Rule is made of
  - property
  - value

```css
p{
  text-align: center;
  color: black;
  font-family: arial
}

h1,h2,h3{
  color: red
}
```
CSS Pattern Matching: Selectors

• **Match things** in a document to apply a rule
  – Document elements
  – Elements with specific ids
  – Element with specific classes

• **More than one pattern can be associated with a rule**
  – Separated with comma

```css
h1, h2, h3
{
  color: red
}
```
CSS2 Selector Patterns

• Pattern matching rules determine which style rules apply to elements in the document tree.
  – Patterns are called selectors that range from simple element names to rich contextual patterns.
  – If all conditions in the pattern are true for a certain element, the selector matches the element.

• Some examples of selectors
  – Type Selectors
  – Class and ID Selectors
  – Descendant and Child Selector
  – Universal Selector
  – Adjacent Selectors
  – Attribute Selectors

See: http://www.w3.org/TR/CSS2/selector.html#q2
Type Selectors

- Matches the name of a document(html) element type
  - The following rule matches all H1 elements in the document tree:
  - `h1 { font-family: sans-serif }`

```html
<style type="text/css">
p{
  text-align: left;
  color:"red";
  font-size: 20px;
}
</style>
...
<p>This is first paragraph </p>
<p>This is second paragraph </p>
```

This is the first paragraph
This is the second paragraph
Class Selectors

- **Match all elements with the given class attribute**
  - Specified with ‘.’ before the class name
  - Only one class attribute can be specified per HTML element

- **Examples**
  - `p.article` - All paragraphs with a class of “article”
  - `.error` - Any element with a class of “error”.

```css
p.first{
  text-align: left;
  color: "red";
  font-size: 20px;
}
p.second{
  text-align: left;
  color: "blue";
  font-size: 16px;
}
...
<p class="first">This is first paragraph</p>
<p class="second">This is second paragraph</p>
```
ID Selectors

- **Matches the given id attribute**
  - An id must be unique in a page.
  - Use a # in the selector

- **Examples**
  - `div#menu` - *selects the div element with the id of “menu”*
  - `#header` - *selects the element with the id of “header”.*

```css
#redtext{
  text-align: left;
  color: "red";
  font-size: 20px;
}
```

```html
<p id="redtext">This is first paragraph</p>
```

Another page

```html
...<h1 id="redtext">This is a header</h1>
```
Descendant Selectors

• Match an element that is the descendant of another element in the document tree

Examples:
• body p {font-weight:bold;}
  – Any paragraph text which is a descendant of body
• tr td p {color: red;}

```
html
  head
  body
  h1
  table
  tr
    td
    p
    tr
    td
    p
```
Child Selector

- Matches when an element is the child of another element

```html
tr > td > p
{
    color: green;
}
```
Adjacent Selectors

- Selects an element that follows another element
  - Text between tags have no effect
- Example:
  - h1 + table { width: 100%; }
Universal Selectors

• Matches an element that is a grandchild or later descendant of another element.
  – Selects paragraphs that are at least one selector removed
  – Note spaces before and after *

• div * p
  – p element that is a grandchild or later descendant of a div
Attribute Selectors

- Attribute selectors may match in four ways:
  - [att]
    - The "att" attribute is set, whatever the value of the attribute.
  - [att=val]
    - "att" attribute value is exactly "val"
  - [att~]=val]
    - "att" attribute value is a space-separated list of "words", one of which is exactly "val"
  - [att|=val]
    - "att" attribute value is a hyphen-separated list of "words", beginning with "val"
      * This is primarily intended to allow language subcode matches (e.g., the "lang" attribute in HTML)

http://www.w3.org/TR/CSS2/selector.html#attribute-selectors
Getting documents ready for CSS

- **CSS is case sensitive:**
  - HTML names should match the name of the selector exactly.
  - `<p class="red" />` does not match `p.Red{}`

- **Use ids and class attributes to mark elements**
  - No spaces
  - `<input id="first-name" />`
  - `<input id="last-name" />`
Inserting a style sheet

• Three ways of inserting a style sheet
  – External Style Sheet
  – Internal Style Sheet
  – Inline Styles
External Style Sheet

• An external style sheet is ideal
  – when the style is applied to many pages
• Link to the style sheet using the <link> tag.
  – The <link> tag goes inside the head section
• Style sheet file
  – should be saved with a .css extension
  – should not contain any html tags

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
<body>
<p class="first">This is first paragraph</p>
<p class="second">This is second paragraph</p>
</body>
```
Internal Style Sheet

- **Internal Styles**
  - should be used when a single document has a unique style
  - Is defined by using `<style>` tag in the head section

```html
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />

  <style type="text/css">
    p {color: white; }
    body {background-color: black; }
  </style>

</head>
```
Inline Styles

- Placing CSS in the HTML code
- This method should be used sparingly
  - For example, when a style is applied to a single occurrence of an element.

```html
<p style="background: black; color: white;">
This is new background and font color with inline CSS
</p>
```
Cascading Order

• Styles will "cascade" by the following rules

  – Browser default
  – External Style Sheet
  – Internal Style Sheet
  – Inline Style
CSS Background

• Defines the background effects on an element

  – background
    • all background properties in one declaration.
  – background-attachment
    • sets whether a background image is fixed or scrolls with the rest of the page.
  – background-color
    • background color of an element
  – background-image
    • Sets an image as the background
  – background-position
    • sets the starting position of a background image
  – background-repeat
    • sets if/how a background image will be repeated
CSS Background Examples

```

h4 { background-color: white; }

body
{
background-image: url(point.gif);
background-repeat: repeat-x
}

p { background-image: url(smallPic.jpg); }

body
{
background-image: url(stars.gif);
background-attachment: scroll
}
```
CSS Text

• Defines the spacing, decoration, and alignment of text
• Properties
  – color
  – direction
  – letter-spacing
  – text-align
  – text-indent
  – text-decoration
  – white-space
  – word-spacing

h2 { text-decoration: underline; }

p { text-indent: 20px; }
CSS Font

- Defines the font in text
- Properties
  - font
  - font-family
  - font-size
  - font-style
  - font-weight
  - ...

```css
p { font: italic small-caps bold 12px arial }
p { font-size: 12px; }
ol { font-size: 10px; }
p { font-style: italic; }
ul { font-weight: bolder; }
```
CSS Border

- Allows for complete customization of the border that appear around HTML elements

- Properties
  - border
  - border-color
  - border-style
  - border-bottom
  - border-bottom-color
  - border-bottom-style
  - border-bottom-width
  - ....

```css
table {
  border-width: 7px;
  border-style: outset; }

td {
  border-width: medium;
  border-style: outset; }

p {
  border-width: thick;
  border-style: solid; }
```
CSS Margin

- Defines the space around the elements
- Properties
  - margin
  - margin-bottom
  - margin-left
  - margin-right
  - margin-top

```css
h5 { margin-top: 0px;
  margin-right: 10px;
  margin-bottom: 10px;
  margin-left: 10px;
  border: 3px solid blue; }
```

This is my header line
CSS and Tableless Layouts

- You can use CSS to do tableless layouts
  - float
  - Position: fixed (position absolute)
  - HTML <div> tags
DIV Based Page Layout with CSS

- Table-based layouts are common
- Use div tags and CSS
  - Reduces markup code
  - Separates content from its visual presentation
- DIV tag
  - Used as a container within our Web page
  - Creating sections or divisions
<body>
  <div id="headerregion"></div>
  <div id="middleregion">
    <div id="sidebar"></div>
    <div id="middle"></div>
  </div>
  <div id="footerregion"></div>
</body>
Liquid Page Designs

• Fixed Locations (position)

```html
<cfhtml>
  <head>
    <style>
      div#headerregion {
        position: absolute;
        width: 100%;
        top: 0;
        left: 0;
        height: 50px;
      }
      /* position:fixed for modern browsers (IE 7 / Firefox) NO scroll */
      body > div#headerregion {
        position: fixed;
      }
    </style>
  </head>
  <body>
    <div id="headerregion"></div>
    <div id="sidebar">
      width: 180px;
      float: left;
    </div>
  </body>
</cfhtml>
```

• Flow around (float)
What You Have Learned

• Cascading Style Sheets are a way to control the look and feel of your HTML documents in an organized and efficient manner.

• With CSS you will be able to
  – Add new looks to your old HTML
  – Completely restyle a web site with only a few changes to CSS code
Hands-On Lab

• Create a CSS to manage look-and-feel of a site
• Manage Layout using `<div>` regions instead of tables
XSLT

Module Road Map

- Web Standards
- Web Architecture: Resources, URI and HTTP
- HTML and XHTML
- XML and XML Schemas
- CSS
- XSLT
XSLT

• **Extensible Stylesheet Language Transformations**
• Transform XML documents into:
  – XML, XHTML, HTML, ..
• Generate *an* output from *two* input files:
  – Content: An XML document
  – Transformation: An XSL document that contains the “template” and XSL transformations to insert content from XML
• **XSL is a *programming language***
  – **NOT** a simple one
  – Debugging your XSL
XSL - Hello World

• **XML:** helloworld.xml

```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="helloworld.xsl"?>
<message>Hello World!</message>
```

• **XSL:** helloworld.xsl:

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- one rule, to transform the input root (/) -->
<xsl:output method="html" />
<xsl:template match="/">
  <html>
    <body>
      <h1>
        <xsl:value-of select="message" />
      </h1>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```

**Result file:**

```html
<html>
  <body>
    <h1>
      Hello World!
    </h1>
  </body>
</html>
```
Anatomy of the XSL file

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="html" />
    <xsl:template match="/">
        <html>
            <body>
                <h1>
                    <xsl:value-of select="message" />
                </h1>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```
How did we get to text in the message?

**Templates**

```xml
<xsl:template match="/">
    <html>
        <body>
            <h1>
                <xsl:value-of select="message" />
            </h1>
        </body>
    </html>
</xsl:template>
```

- **Alternative select statements**
  - ./message
  - with XPath functions
    - /message/text()
    - ./message/text()
Inside the XSLT Transformation

1. Read the XML document and store it as a Tree of nodes

3. Match templates to parts of the tree
   – `<xsl:template match="/"> select the entire tree
   – `<xsl:template match=".""> use it to select subsets

4. Apply the rules in each the template to create a new structure
   – `<xsl:apply-templates/> Call additional templates from the root template

5. Unmatched parts of the XML tree are not changed

7. Write the transformed tree as a text document
XSL can run on the server and the client

**Server:**
- Xalan, Saxon, Xerces, etc. can be used to read and write files
- Use XSLT to change XML files into HTML files before sending them to the client
- More portable (Less to expect from a browser)

**Client**
- A *modern* browser can use XSLT to change XML into HTML on the client side
- Internet Explorer 6+
- Netscape 6+
- Mozilla, Firefox 1+, Opera 8+, ..
xsl:value-of

```xml
<xsl:value-of select="XPath expression"/>
```

* selects the contents of an element and adds it to the output stream
  - The select attribute is required
  - Notice that xsl:value-of is not a container, hence it needs to end with a slash

* Example:
  ```xml
  <h1> <xsl:value-of select="message"/> </h1>
  ```
xsl:for-each

Loop statement
<xsl:for-each select="XPath expression">
   Text to insert and rules to apply
</xsl:for-each>

• Example: Select all books (//book) and list the titles (title):
   <ul>
      <xsl:for-each select="/book">
         <li><xsl:value-of select="title"/></li>
      </xsl:for-each>
   </ul>
Filtering output

Filter output with a criterion

title[../genre='mystery']

Legal filter operators are:

=  !=  <  >

Example: Select all school books (/book) and list the titles (title):

```xml
<ul>
  <xsl:for-each select="/book">
    <li>
      <xsl:value-of select="title[../genre='mystery']"/>
    </li>
  </xsl:for-each>
</ul>
```

*title and genre are at the same level of the XML tree (they are both inside the book). “../" takes us to the level of the book and we select “genre”*

There is a catch!

Other items will also show in the list as empty items
But it doesn’t work right!

```xml
<xsl:for-each select="//book">
  <li>
    <xsl:value-of select="title/../genre='mystery'"/>
  </li>
</xsl:for-each>
```

outputs for every book,
- Empty bullets for other books
- Do not use `xsl:value-of` to filter

Alternative Filter:
```xml
<xsl:for-each select="//book[./genre='mystery']">
  <li>
    <xsl:value-of select="title"/>
  </li>
</xsl:for-each>
```
xsl:if

- Include content when condition is true
- Example:

```xml
<xsl:for-each select="/book">
    <xsl:if test="genre='mystery'">
        <li>
            <xsl:value-of select="title"/>
        </li>
    </xsl:if>
</xsl:for-each>
```
xsl:choose

- XSL switch ... case ... default statement
- The syntax is:

  ```xml
  <xsl:choose>
    <xsl:when test="some condition">
      ... some code ...
    </xsl:when>
    <xsl:otherwise>
      ... some code ...
    </xsl:otherwise>
  </xsl:choose>
  ```
**xsl:sort**

- **Sorting inside an **xsl:for-each****: Attribute of the sort tells what field to sort on

**Example:**

```xml
<ul>
  <xsl:for-each select="//book">
    <xsl:sort select="author"/>
  </xsl:for-each>
</ul>
```
• `<xsl:text>...</xsl:text> helps with:
  – Whitespaces and special characters

<xsl:text disable-output-escaping="yes">&amp;nbsp;</xsl:text>
Creating tags from XML data

• XML
  <label>Eteration A.S.</label>
  <url>http://www.eteration.com</url>

• Desired Result
  <a href="http://www.eteration.com">
    Eteration A.S.
  </a>

• We cannot use invalid XML within XSL
  – <xsl-valueof> does not work inside a tag
  – Same with <img /> tags
Solutions

Using: `<xsl:attribute name="...">

`<a>`

  `<xsl:attribute name="href">
   `<xsl:value-of select="url"/>
  </xsl:attribute>

  `<xsl:value-of select="label"/>

`</a>`

Using attribute value template: `{...}

`<a href="{url}"`

  `<xsl:value-of select="label"/>

`</a>`
Modularization with Templates

- XSL can be divided into multiple templates using:
  - Call by name
    \[
    \text{<xsl:call-template name="template_name"/>}\]
  - By using XML tree select statements:
    - \[
        \text{<xsl:apply-templates select="book"/>}\]

```xml
<xsl:template match="/">
  <html>
    <body>
      <xsl:apply-templates />
    </body>
  </html>
</xsl:template>
<xsl:template match="book">
  <h1>Book Information</h1>
  <xsl:apply-templates select="title" />
</xsl:template>
```
xsl:apply-templates

- **Apply template rule**
  - current element
  - current element’s child nodes

- **Optional: select attribute,**
  - Applies the template rule only to the child that matches

- **Multiple <xsl:apply-templates>**
  - Select attributes
  - the child nodes are processed in the same order as the
    <xsl:apply-templates> elements
When templates are ignored

- A template is skipped if it does not apply
- Use `select="/"` to always process
  - If it didn’t, nothing would ever happen

Warning:
If a template applies to an element, templates are not automatically applied to its children
Applying templates to children

```
<book>
  <title>Les Miserables</title>
  <author>Victor Hugo</author>
</book>
```

With apply-template line: 
Les Miserables by *Victor Hugo*

Without apply-template line: 
Les Miserables

```
<xsl:template match="/">
  <html>
    <head></head>
    <body>
      <b><xsl:value-of select="/book/title" /></b>
    </body>
    <xsl:apply-templates select="/book/author" />
  </body>
</html>
</xsl:template>

<xsl:template match="/book/author">
  by <i><xsl:value-of select="." /></i>
</xsl:template>
```
Calling named templates

- You can name a template, then call it, similar to the way you would call a method in Java.
- The named template:
  
  ```xml
  <xsl:template name="myTemplateName">
    ...body of template...
  </xsl:template>
  ```

- A call to the template:
  
  ```xml
  <xsl:call-template name="myTemplateName"/>
  ```

- Or:
  
  ```xml
  <xsl:call-template name="myTemplateName">
    ...parameters...
  </xsl:call-template>
  ```
Templates with parameters

```xml
<xsl:call-template name="showPeople">
  <xsl:with-param name="title" select="/project/title"/>
  <xsl:with-param name="people" select="/project/team/members"/>
</xsl:call-template>

• Parameterized template:
  <xsl:template name="showPeople">
    <xsl:param name="title"/>
    <xsl:param name="people"/>
    ...template body...refer to parameters as "$title" and "$people"
  </xsl:template>
  – Parameters are matched up by name, not by position
```
Generating XSL output with Java

Basic procedure for XSL transformation with Xalan:

2. Instantiate a TransformerFactory
   - Use the TransformerFactory static newInstance()

3. Generate a Transformer from XSLT source
   - TransformerFactory newTransformer(Source stylesheet)
   - Template

4. Apply transformation
   - transform(Source xmlSource, Result transformResult)
   - The Templates object to the XML Source
What You Have Learned

• XSL and XSL constructs
• Transforming XML document into different types of documents
Hands-On Lab

- Create an XSLT to create the Web page from XML
  - objectshop.xml
  - objectshop.xsl
- Use CSS to create the presentation
  - objectshop.css