XML

Objectives

- Define XML and its relationship to HTML
- Examine uses of XML
- Describe the syntax of XML
- Describe the processing of XML documents
- Show uses of XML in Web Services

Introduction to XML

Introduction

HTML makes it easy to identify meaning of text data

HTML Markup identifies…
tables content,
lists,
images,
…

Easy to parse, easy to read, standard format, but…
HTML is for web pages

Introduction

HTML was developed using SGML in the early 1990s

SGML is an example of a ***meta-markup language***

- Developed in the early 1980s; ISO std. In 1986

HTML has a fixed set of tags - specifically for Web docs

Introduction

It would be nice to define our own tags with their own meanings

Problem with using SGML:

- It's too large and complex to use, and it is very difficult to build a parser for it
- Solution: **XML**, a "lite" version of SGML

Introduction

eXtensible Markup Language (XML) is a meta-markup language that can be used to **define markup languages** that can define the meaning of specific kinds of information

XML is not a replacement for HTML

XML can be (and was) used to redefine HTML

The result was XHTML, but this was not the major purpose of XML

Introduction

XML provides a very simple and universal way of **storing** and **transferring data** of any kind

Does not predefine any tags, but always for specifying them

Has no hidden specifications

Documents can be parsed with a single (standard) **parser**

Definitions

An XML-based markup language is a ***tag set***
(technically *XML application*, but that term is confusing)

A document that uses an XML-based markup language is an ***XML document***

An ***XML processor*** is a program that parses XML documents and provides the parts to an application

Uses of XML

Some examples of using XML:

Common Data Format (CDF) – for describing and storing scalar and multidimensional data

Scalable Vector Graphics (SVG) – to describe vector images

Mathematics Markup Language (MathML) – to integrate mathematical notation into a Web document

Chemical Markup Language (CML) - to support chemistry

GPS eXchange Format (GPX) – to describe GPS data

Medical Markup Language (MML) – to represent medical information

Office Open XML (OOXML) – for Microsoft Office

XML Syntax

XML Syntax

Two distinct levels of syntax:

- General low-level rules that apply to all XML documents
- For a particular XML tag set, either a ***document type definition (DTD)*** or an ***XML schema***

General XML Syntax

Same as those of XHTML

XML documents consist of

data elements

markup declarations: instructions for the XML parser

processing instructions: for the application program that is processing the data in the document

All XML documents begin with an XML declaration:

```
<?xml version = "1.0" encoding = "utf-8"?>
```

General XML Syntax

Markup tags and other XML **names** have to:

- Begin with a letter or an underscore
- Can include digits, hyphens, and periods
- Have no length limitation
- Are **case sensitive** (unlike HTML names)

Every XML document defines a single ***root element***, whose opening tag **must appear as the first line of the document**

General XML Syntax

An XML document that follows all of these rules is ***well formed***

Example XML document:

```
<?xml version = "1.0" encoding = "utf-8" ?>
<ad>
  <year> 1960 </year>
  <make> Cessna </make>
  <model> Centurian </model>
  <color> Yellow with white trim </color>
  <location>
    <city> Gulfport </city>
    <state> Mississippi </state>
  </location>
</ad>
```

Attributes

Just like in HTML, tags can have attributes, but…
attributes are not used in XML the way they are in HTML

In XML, you often define a **new nested tag** to provide more info about the content of a tag

Nested tags are better than attributes, because attributes cannot describe structure and the structural complexity may grow

Attributes should always be used to identify numbers or names of elements (like HTML `id` and `name` attributes)

Attributes

```
<!-- A tag with one attribute -->
<patient name = "Maggie Dee Magpie">
   ...
</patient>


<!-- A tag with one nested tag -->
<patient>
   <name> Maggie Dee Magpie </name>
   ...
</patient>
```

Attributes

```
<!-- A tag with one nested tag, which contains
       three nested tags -->
<patient>
   <name>
      <first> Maggie </first>
      <middle> Dee </middle>
      <last> Magpie </last>
   </name>
   ...
</patient>
```

Entities

XML documents often consist of one or more ***entities***

- Entities range from a single special character to a book chapter
- An XML document has one ***document entity***

Reasons for using entities:

- Large documents are easier to manage
- Repeated entities need not be literally repeated
- ***Binary entities*** can only be referenced in the document entities (XML is all text!)

Entities

Entity names

- No length limitation
- Must begin with a letter, a dash, or a colon
- Can include letters, digits, periods, dashes, underscores, or colons

A reference to an entity has the form:

&entity_name;

Predefined entities (as in XHTML):

<	<
>	>
&	&
"	"
'	'

Namespaces

A ***markup vocabulary*** is the collection of all of the element types and attribute names of a markup language (a tag set)

An XML document may define its own tag set and also use those of another tag set - CONFLICTS!

An ***XML namespace*** is a collection of names used in XML documents as element types and attribute names

The name of an XML namespace has the form of a URI

Namespaces

A ***namespace declaration*** has the form:

```
<element_name xmlns[:prefix] = URI>
```

The ***prefix*** is a short name for the namespace, which is attached to names from the namespace in the XML document

```
<gmcars xmlns:gm = "http://www.gm.com/names">
```

In the document, you can use `<gm:pontiac>`

Purposes of the prefix

- Shorthand
- URI includes characters that are illegal in XML

Namespaces

Can declare two namespaces on one element

```
<gmcars xmlns:gm = "http://www.gm.com/names"
xmlns:html = "http://www.w3.org/1999/xhtml">
```

The `gmcars` element can now use `gm` names and xhtml names

One namespace can be made the default by leaving the prefix out of the declaration

Namespaces

Example

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

XML Schemas

XML Schemas

An ***XML schema*** is an XML document that

- Specifies the elements and attributes of an XML language
- Specifies the structure of its instance XML documents
- Specifies the data type of every element and attribute of its instance XML documents

XML Schemas

Schemas are written using a **namespace**:
`http://www.w3.org/2001/XMLSchema`

Every XML schema has a single **root**, `schema`

The `schema` element must specify the namespace for schemas as its `xmlns:xsd` attribute

XML Schemas

Every XML schema itself **defines a tag set**, which must be named, e.g.:

```
targetNamespace =
      "http://cs.uccs.edu/planeSchema"
```

If we want to include nested elements, we must set the `elementFormDefault` attribute to `qualified`

The **default namespace** must also be specified, e.g.:

```
xmlns = "http://cs.uccs.edu/planeSchema"
```

XML Schemas

A complete example of a ***schema element***:

```
<xsd:schema
    <!-- Namespace for the schema itself -->
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"

    <!-- Namespace where elements defined here will
    be placed -->
    targetNamespace =
    "http://cs.uccs.edu/planeSchema"

    <!-- Default namespace for this document -->
    xmlns = "http://cs.uccs.edu/planeSchema"

    <!-- Next, specify non-top-level elements to be
    in the target namespace -->
    elementFormDefault = "qualified">
```

Defining a Schema Instance

When defining an **instance document,** the ***root element*** must specify the namespaces it uses:

- The default namespace
- The standard namespace for instances (`XMLSchema-instance`)
- The location where the default namespace is defined, using the `schemaLocation` attribute

```
<planes
  xmlns = "http://cs.uccs.edu/planeSchema"
  xmlns:xsi =
   http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation =
   "http://cs.uccs.edu/planeSchema/planes.xsd" >
```

Schema Data Types

An XML schema can be used to define types of data (data that is stored in elements):

Element Categories

Simple: strings only, no attributes and no nested elements

Complex: can have attributes and nested elements

Schema Data Types

XMLS defines 44 data types

Primitive: `string`, `Boolean`, `float`, ...

Derived: `byte`, `decimal`, `positiveInteger`, ...

User-defined (derived) data types – specify constraints on an existing type (the base type)

- Constraints are given in terms of ***facets*** (`totalDigits`, `maxInclusive`, etc.)

Schema Data Types

Both simple and complex types can be either ***named*** or ***anonymous***

With XML schema (XMLS), context is essential, and elements can be either:

- ***Local***, which appears inside an element that is a child of `schema`
- ***Global***, which appears as a child of `schema`

Simple Types

To define a **simple type**, use the `element` tag and set the `name` and `type` attributes

```
<xsd:element name = "bird" type = "xsd:string" />
```

An instance could have:

```
<bird> Yellow-bellied sap sucker </bird>
```

Element values can be constant, specified with the `fixed` attribute

```
fixed = "three-toed"
```

User-Defined Types

User-defined types are defined in a **simpleType** element using ***facets*** specified in the content of a **restriction** element

Facet values are specified with the **value** attribute

```
<xsd:simpleType name = "middleName" >
    <xsd:restriction base = "xsd:string" >
        <xsd:maxLength value = "20" />
    </xsd:restriction>
</xsd:simpleType>
```

Complex Types

There are several categories of complex types, but we discuss just one:

element-only elements

(have elements in their context but no text)

Element-only elements are defined with the `complexType` element

- Use the **`sequence`** tag for nested elements that must be in a particular order
- Use the **`all`** tag if the order is not important

Complex Types

Example

```
<xsd:complexType name = "sports_car" >
    <xsd:sequence>
      <xsd:element  name = "make"
                    type = "xsd:string" />
      <xsd:element  name = "model "
                    type = "xsd:string" />
      <xsd:element  name = "engine"
                    type = "xsd:string" />
      <xsd:element  name = "year"
                    type = "xsd:string" />
    </xsd:sequence>
 </xsd:complexType>
```

Nested elements can include attributes that give the allowed number of occurrences (`minOccurs`, `maxOccurs`, `unbounded`)

Complex Types

We can define ***nested elements*** elsewhere

```
<xsd:element  name = "year" >
  <xsd:simpleType>
    <xsd:restriction  base = "xsd:decimal" >
       <xsd:minInclusive  value = "1990" />
       <xsd:maxInclusive  value = "2003" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

We can then reference this ***global element*** in the complex type with the **`ref`** attribute

```
<xsd:element ref = "year" />
```

XML Schema Validation

Validating Instances of XML Schemas

One validation tool is **`xsv`**, which is available from:
http://www.ltg.ed.ac.uk/~ht/xsv-status.html

Note: If the schema is incorrect (bad format), `xsv` reports that it cannot find the schema

Viewing XML Documents

Displaying Raw XML Documents

An XML browser should have a default style sheet for an XML document that does not specify one

You get a stylized listing of the XML

EXAMPLE: **planes1.xml**

Displaying XML Documents with CSS

A **CSS style sheet for an XML document** is just a list of its **tags** and **associated styles**

The connection of an XML document and its style sheet is made through an `xml-stylesheet` processing instruction

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

EXAMPLE: **planes.xml (uses planes.css)**

Using XSLT

XSLT Style Sheets

The general method for controlling the presentation of XML documents is using **XSL**

XSL is the ***EXtensible Stylesheet Language***

Split into three parts:

- **XSLT** – Transformations
- **XPATH** - XML Path Language
- **XSL-FO** - Formatting objects

XSLT uses style sheets to specify **transformations**

XSL Transformations

Transformations are applied by an **XSLT processor.**

An ***XSLT processor*** merges an **XML document** into an **XSLT document (a style sheet)** to create an **XSL document**

This merging is a ***template-driven process***

XSL Transformations

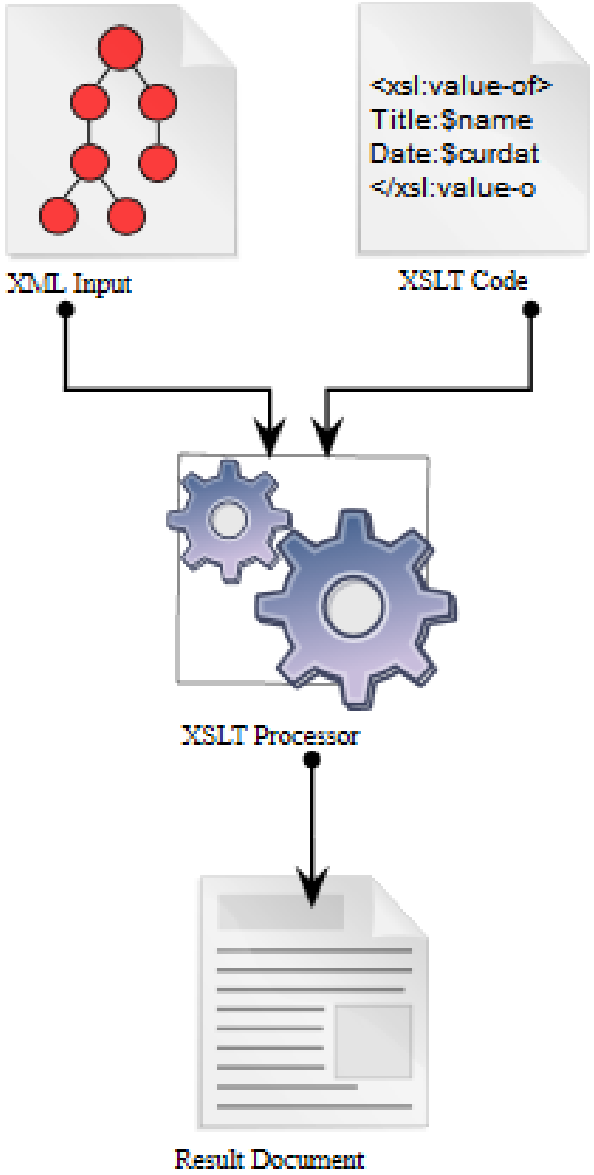
The process works as follows:

- XSLT processor examines the nodes of the XML document, comparing them with the ***XSLT templates***

- Matching templates are put in a **list of templates** that could be applied– if more than one, a set of rules determine which is used (only one is applied)

- Applying a template causes its body to be placed in the ***XSL document***

XSLT Style Sheets



XSL Transformations

An XSLT style sheet can specify page layout, page orientation, writing direction, margins, page numbering, etc.

The processing instruction we used for connecting a **CSS** style sheet to an XML document is also used to connect an **XSLT** style sheet to an XML document:

```
<?xml-stylesheet type = "text/xsl"
                 href = "XSLT style sheet"?>
```

XSLT Style Sheets

An ***XSLT style sheet*** is an XML document with a single element, **`stylesheet`**, which defines namespaces

```
<xsl:stylesheet xmlns:xsl =
      "http://www.w3.org/1999/XSL/Format"
   xmlns = "http://www.w3.org/1999/xhtml">
```

If a style sheet matches the **root** element of the XML document, it is matched with the ***template***:

```
<xsl:template match = "/">
```

XSLT Style Sheets

XSLT documents can include two different kinds of elements:

- those with content and
- those for which the content will be merged from the XML document

Elements with content often represent **HTML elements**

XSLT elements that represent HTML elements are **simply copied** to the merged document

XSLT Style Sheets - Content Only Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
              xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td>.</td>
      <td>.</td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

XML Transformation

To get content (data) form an XML document, we need to use the `value-of` element

The XSLT **`value-of`** element

- Has no content
- Uses a **`select`** attribute to specify part of the XML data to be **merged into the new document**
- The value of `select` can be any branch of the **document tree**

Example:

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

XSLT Style Sheets - Merging Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <tr>
      <td><xsl:value-of select="catalog/cd/title"/></td>
      <td><xsl:value-of select="catalog/cd/artist"/></td>
    </tr>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

XSLT for-each element

The XSLT **`for-each`** element is used when an XML document has a sequence of the same elements

Example use:

For each CD, output the artist and title

XSLT for-each Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
  <body>
  <h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32">
      <th>Title</th>
      <th>Artist</th>
    </tr>
    <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
    <tr>
      <td><xsl:value-of select="title"/></td>
      <td><xsl:value-of select="artist"/></td>
    </tr>
    </xsl:for-each>
  </table>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

Examples

xslplane1.xsl

xslplane.xml

xslplane2.xsl

xslplanes.xml

xslplanes.xsl

Parsing XML

XML Processors

Job of the ***XML processor*** (aka parser):

- **Check the syntax** of a document for well-formedness
- Replace all references to **entities** by their definitions
- Copy **default values** (from DTDs or schemas) into the document
- If a DTD or schema is specified and the processor includes a validating parser, the structure of the document is **validated**

Two ways to check well-formedness:

- A browser with an XML parser
- A stand-alone XML parser

XML Processors

There are two different approaches to designing XML processors: SAX and DOM approaches

The ***SAX (Simple API for XML)*** Approach:

- Widely accepted and supported
- Based on the concept of *event processing*:
- Every time a syntactic structure (e.g., a tag, an attribute, etc.) is recognized, the **processor** raises an event
- The **application** defines event handlers to respond to the syntactic structures

DOM Approach

The alternative to SAX is to use a DOM processor

The ***DOM processor*** builds a DOM **tree structure** of the document (Similar to the processing by a browser of an XHTML document)

When the tree is complete, it can be **traversed** and **processed**

DOM Approach

Advantages of the DOM approach:

1. Good if any part of the document must be **accessed more than once**
2. If any rearrangement of the document must be done, it is facilitated by having a representation of **the whole document in memory**
3. A **random access** to any part of the document is possible
4. Because the whole document is parsed before any processing takes place, **processing of an invalid document is avoided**

DOM Approach

Disadvantages of the DOM approach:

1. Large documents **require a large memory**

2. The DOM approach is **slower**

Web Services

Web Services

The Web began as provider of markup documents, served through the HTTP methods, GET and POST

- An **information service system**

A ***Web Service*** is closely related to an information service

The ultimate goal of Web services:

Allow **different software** in **different places**, written in **different languages** and resident on **different platforms**, to **connect** and **interoperate**

Web Services

The original Web services were provided via ***Remote Procedure Call (RPC)***, through two technologies, **DCOM** and **CORBA**

DCOM and CORBA use different protocols, which defeats the goal of universal component interoperability

Web Services

There are three roles required to provide and use Web services:

1. Service providers
2. Service requestors
3. A service registry

Web Services use several technologies:
XML, WSDL, UDDI, SOAP

XML is used for storing and transmitting data

Web Services

Web Service Definition Language (WSDL) is used to describe available services, as well as message protocols for their use

Universal Description, Discovery, and Integration Service (UDDI) is used to create Web services registry, and also methods that allow a remote system to determine which services are available

WSDL Example

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

Web Services

Standard Object Access Protocol (SOAP) is an XML-based specification that defines the forms of messages and RPCs

- Supports the exchange of information among distributed systems
- A SOAP message is an XML document that includes an envelope
- The body of a SOAP message is either a request or a response

SOAP Request Example

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

SOAP Response Example

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Web Services

Example of a Web Service

National Digital Forecast Database (NDFD)

Simple Object Access Protocol (SOAP)

Web Service

http://graphical.weather.gov/xml/

http://graphical.weather.gov/xml/SOAP_server/ndfdXML.htm

Summary

- Extensible Markup Language (XML) is a meta-markup language that can be used to define markup languages
- XML provides a very simple and universal way of storing and transferring data of any kind
- An XML namespace is a collection of names used in XML documents
- XML schema is a language used to define XML namespaces
- The general method for controlling the presentation of XML documents is using XSL

Summary (cont.)

- XSL uses XSLT to define transformations
- An XSLT processor merges an XML document into an XSLT document (a style sheet) to create an XSL document
- There are two different approaches to designing XML processors: SAX and DOM approaches
- Web Services allow different software in different places, written in different languages and resident on different platforms, to connect and interoperate
- Web Services use several technologies: XML, WSDL, UDDI, SOAP