7.1 Introduction

- SGML is a meta-markup language

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

- HTML was developed using SGML in the early 1990s - specifically for Web documents

  - *Two problems with HTML*:

    1. Fixed set of tags and attributes

       - User cannot define new tags or attributes

       - So, the given tags must fit every kind of document, and the tags cannot connote any particular meaning

    2. There are few restrictions on arrangement or order of tag appearance in a document

- One solution to the first of these problems: Let each group of users define its own tags (with implied meanings)

  (i.e., design their own "HTML"s using SGML)

7.1 Introduction (continued)

- *Problem with using SGML*:

  - It's too large and complex to use, and it is very difficult to build a parser for it

- *A better solution*: Define a lite version of SGML

- XML is not a replacement for HTML

  - HTML is a markup language used to describe the layout of any kind of information

  - 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 a very simple and universal way of storing and transferring data of any kind

- XML does not predefine any tags

- XML has no hidden specifications

7.1 Introduction (continued)

- We will refer to an XML-based markup language as a *tag set*

- Strictly speaking, a tag set is an *XML application*, but that terminology can be confusing

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

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

7.2 Uses of XML

- *Examples:*

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

  Scalable Vector Graphics (SVG) – to describe vector images

7.2 Uses of XML (continued)

- *Examples:*

  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

7.3 The Syntax of XML

- The syntax of XML is in two distinct levels:

  1. The general low-level rules that apply to all XML documents

  2. An XML schema for a particular XML tag set

7.3 The Syntax of XML (continued)

- *General XML Syntax*

- XML documents consist of:

  1. data elements
  2. markup declarations (instructions for the XML parser)
  3. 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"?>
```

- *XML names*:

  - Must begin with a letter or an underscore
  - They can include digits, hyphens, and periods
  - There is no length limitation
  - They are case sensitive (unlike HTML names)

7.3 The Syntax of XML (continued)

- *Syntax rules for XML*: same as those of XHTML

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

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

```
<?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>
```

7.3 The Syntax of XML (continued)

- 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)

7.3 The Syntax of XML (continued)

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


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


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

7.4 XML Document Structure

- An XML document often uses two auxiliary files:
  - One to specify the structural syntactic rules
  - One to provide a style specification
- An XML document has a single root element, but often consists of one or more entities
- An XML document has one *document entity*

- *Reasons for entity structure*:
  1. Large documents are easier to manage
  2. Repeated entities need not be literally repeated
  3. Binary entities can only be referenced in the document entities

7.4 XML Document Structure (continued)

- When the XML parser encounters a reference to a non-binary entity, the entity is merged in

- *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 HTML):

<	<
>	>
&	&
"	"
'	'

7.4 XML Document Structure (continued)

- If several predefined entities must appear near each other in a document, it is better to avoid using entity references

  - Character data section

    ```
    <![CDATA[ content ]]>
    ```

    e.g., instead of

    ```
    Start > > > > HERE
                    < < < <
    ```

    use

    ```
    <![CDATA[Start >>>> HERE <<<<]]>
    ```

  - If the `CDATA` content has an entity reference, it is taken literally

7.5 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 URL

  - A namespace declaration has the form:

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

    - 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*:
  1. Shorthand
  2. URL includes characters that are illegal in XML

7.5 Namespaces (continued)

- 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 html/xhtml names

7.6 XML Schemas

- *Three purposes of an XML schema*:

1. Specify the elements and attributes of an XML language
2. Specify the structure of its instance XML documents
3. Specify the data type of every element and attribute of its instance XML documents

- 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

- Every XML schema itself defines a tag set, which must be named

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

7.6 XML Schemas (continued)

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

- The default namespace must also be specified

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

- 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">
```


7.6 XML Schemas (continued)

- *Defining an instance document*

  - The root element must specify the namespaces it uses

    1. The default namespace

    2. The standard namespace for instances (`XMLSchema-instance`)

    3. The location where the default namespace is defined, using the `schemaLocation` attribute, which is assigned two values

```
<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" >
```

- *Data Type Categories*

  1. Simple (strings only, no attributes and no nested elements)

  2. Complex (can have attributes and nested elements)

7.6 XML Schemas (continued)

- 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.)
- Both simple and complex types can be either named or anonymous
- Elements can be either:
  1. Local, which appears inside an element that is a child of `schema`, or
  2. Global, which appears as a child of `schema`

7.6 XML Schemas (continued)

- *Defining 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*
  - Defined in a `simpleType` element, using facets specified in the content of a `restriction` element
    - Facet values are specified with the `value` attribute

7.6 XML Schemas (continued)

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

- There are several c*ategories* of complex types, but we discuss just one, *element-only elements*

  - 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

7.6 XML Schemas (continued)

```
<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`)

  → SHOW `planes.xsd` and `planes1.xml`

- The `choice` element can have any number of elements, only one of which can appear

- 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>
```

7.6 XML Schemas (continued)

- The global element can be referenced in the complex type with the `ref` attribute

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

- *Entities in schemas*

  - Defined as an element:

```
<xsd:element name = "c" type = "xsd:token"
             fixed = "Cessna" />
```

  Use: `<make> <c> </make>`

- *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

7.7 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

→ SHOW `planes.xml` with a browser

7.8 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"?>
```

→ SHOW `planes.css` and run `planes.xml`

7.9 XSLT Style Sheets

- XSL began as a standard for presentations of XML documents
  - Split into three parts:
    - XSLT – Transformations
    - XPATH - XML Path Language
    - XSL-FO - Formatting objects for printable docs
  - XSLT uses style sheets to specify transformations

7.9 XSLT Style Sheets (continued)

- 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
  - 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
- An XSLT style sheet can specify page layout, page orientation, writing direction, margins, page numbering, etc.
- The processing instruction we used for connecting a XSLT style sheet to an XML document is used to connect an XSLT style sheet to an XML document

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

7.9 XSLT Style Sheets (continued)

```
<?xml version = "1.0" encoding = "utf-8" ?>
   <!-- xslplane.xml -->
   <?xml-stylesheet type = "text/xsl"
                    href = "xslplane1.xsl" ?>
   <plane>
      <year> 1977 </year>
      <make> Cessna </make>
      <model> Skyhawk </model>
      <color> Light blue and white </color>
   </plane>
```

- 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 documents include two different kinds of elements, those with content and those for which the content will be merged from the XML doc
  - Elements with content often represent HTML elements

```
<span style = "font-size: 14">
    Happy Easter!
</span>
```


7.9 XML Transformations and Style Sheets (continued)

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

- 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

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

  - The value of `select` can be any branch of the document tree

→ SHOW `xslplane1.xsl` and display `xslplane.xml`

- `xslplane1.xsl` is more complex than necessary

→ SHOW `xslplane2.xsl`

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

→ SHOW `xslplanes.xml`, `xslplanes.xsl` & display

7.10 XML Processors

- ***Purposes*:**

  1. Check the syntax of a document for well-formedness

  2. Replace all references to entities by their definitions

  3. Copy default values (from XML schema) into the document

  4. If an XML schema is specified and the processor includes a validating parser, the structure of the document is validated

- ***Two ways to check well-formedness:***

  1. A browser with an XML parser

  2. A stand-alone XML parser

- **There are two different approaches to designing XML processors:**

  - **SAX and the DOM approach**

7.10 XML Processors (continued)

- ***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**

- ***The DOM Approach***

  - **The DOM processor builds a DOM tree structure of the document**

    (Similar to the processing by a browser of an HTML document)

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

7.10 XML Processors (continued)

- *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. 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

- *Disadvantages of the DOM approach*:

  1. Large documents require a large memory

  2. The DOM approach is slower

- Note: Most DOM processors use a SAX front end

7.10 Web Services

- 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

- 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 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

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

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

7.10 Web Services (continued)

- *Web Service Definition Language* (WSDL)
  - Used to describe available services, as well as message protocols for their use
- *Universal Description, Discovery, and Integration Service* (UDDI)
  - Used to create Web services registry, and also methods that allow a remote system to determine which services are available
- *Standard Object Access Protocol* (SOAP)
  - 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

