

8.1 Introduction

- SGML is a **meta-markup language**
(note: **markup vs content**)
- 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 no 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 their own tags
(with implied meanings)

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

8.1 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 SGML-lite**)

- 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

- All documents described with an XML-derived markup language can be parsed with a single parser

8.1 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
- XHTML is HTML defined with XML
- Both IE6 and NS6 & NS7/Mozilla/Firefox support basic XML

8.2 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 (**predefined!**)
 2. For a particular XML tag set, either a document type definition (**DTD**) or an **XML schema** (**application specific!**)

8.2 The Syntax of XML (continued)

- General XML Syntax
- XML documents have 1) **data elements**, 2) **markup declarations** (instructions for the XML parser), and 3) **processing instructions** (for the application program that is processing the data in the document)
- All XML documents begin with an XML declaration, (a processing instruction (PI)) enclosed in the delimiters “<?” and “>”:

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

PI's consist of a PI target and a PI value

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

- XML comments are just like HTML comments
- 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)

8.2 The Syntax of XML (continued)

- *Syntax rules for XML*: (similar to those for XHTML)
 - Every XML document defines a **single root** element, whose opening tag must appear as the first line of the document
 - Every element that has content must have a closing tag
 - Tags must be properly nested
 - All attribute values must be quoted (**"double quotes"** or **'single quotes'**)
- An XML document that follows all of these rules is **well formed**

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

8.2 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)

8.2 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>
```

8.3 XML Document Structure

- An XML document often uses **two types of auxiliary files**:
 - Ones to specify the **structural syntactic rules**
 - Ones to provide a **style specification**
- An XML document has a single root element, but often consists of one or more entities
 - Entities range from a single special character (e.g., Unicode) to a book chapter
 - An XML document has one *document entity*
 - All other entities are referenced in the 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 (XML is all text!)

8.3 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;`
- One common use of entities is for special characters that may be used for markup delimiters
 - These are **predefined** (as in XHTML):

<	<
>	>
&	&
"	"
'	'

- The user can only define entities in a DTD

8.3 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

8.4 Data Type Definitions

- A **DTD** is a set of structural rules called *declarations*
 - These rules specify a set of elements, along with how and where they can appear in a document
- Purpose: provide a standard form for a collection of XML documents
- Not all XML documents have or need a DTD
- The DTD for a document can be internal or external
- Errors in DTD: Find them early!
 - All of the declarations of a DTD are enclosed in the block of a `DOCTYPE` markup declaration
- DTD declarations have the form:

```
<!keyword ... >
```
- There are four possible declaration keywords: **ELEMENT**, **ATTLIST**, **ENTITY**, and **NOTATION**

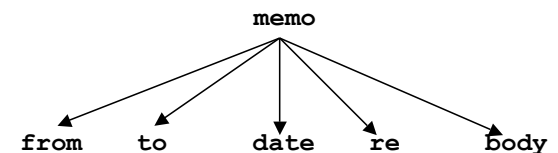
8.4 Data Type Definitions (continued)

- Declaring *Elements*
 - Element declarations are similar to **EBNF** (**extended Backus Naur Form**)
 - An element declaration specifies the names of an element, and the element's structure
 - If the element is a leaf node of the document tree, its structure is in terms of characters
 - If it is an internal node, its structure is a list of children elements (either leaf or internal nodes)
- General form:

```
<!ELEMENT element_name (list of child names) >
```

e.g.,

```
<!ELEMENT memo (from, to, date, re, body) >
```



8.4 Data Type Definitions (continued)

- Declaring Elements (continued)

- Child elements can have modifiers, +, *, ?

e.g.,

```
<!ELEMENT person
    (parent+, age, spouse?, sibling*)>
```

- **Leaf nodes** specify data types, most often **PCDATA**, which is an acronym for parsable character *data*

- Data type could also be **EMPTY** (no content) and **ANY** (can have any content)

- Example of a leaf declaration:

```
<!ELEMENT name (#PCDATA)>
```

- Declaring Attributes

- General form:

```
<!ATTLIST el_name at_name at_type [default]>
```

8.4 Data Type Definitions (continued)

- Declaring Attributes (continued)

- **Attribute types**: there are many possible, but we will consider only **CDATA**

- **Default values**:

a value

#FIXED value (every element will have this value),

#REQUIRED (every instance of the element must have a value specified), or

#IMPLIED (no default value and need not specify a value)

- e.g.,

```
<!ATTLIST car doors CDATA "4">
<!ATTLIST car engine_type CDATA #REQUIRED>
<!ATTLIST car price CDATA #IMPLIED>
<!ATTLIST car make CDATA #FIXED "Ford">
```

```
<car doors = "2" engine_type = "v8">
...
</car>
```

8.4 Data Type Definitions (continued)

- Declaring *Entities*

- Two kinds:

- A *general entity* can be referenced anywhere in the content of an XML document
- A *parameter entity* can be referenced only in a markup declaration [%]

- General form of declaration:

```
<!ENTITY [%] entity_name "entity_value">
```

e.g., `<!ENTITY jfk "John Fitzgerald Kennedy">`

- A reference: `&jfk;`

- If the entity value is longer than a line, define it in a separate file (an *external text entity*)

```
<!ENTITY entity_name SYSTEM "file_location">
```

→ SHOW [planes.dtd](#)

8.4 Data Type Definitions (continued)

- XML Parsers: SAX, DOM, xerces-j, etc.

- Always check for **well formedness**
- Some check for validity, relative to a given DTD
 - Called *validating XML parsers*
- You can download validating XML parsers from:

```
http://xml.apache.org/xerces-j/index.html  
and many other locations ...
```

- Internal DTDs

```
<!DOCTYPE root_name [  
    ...  
>
```

- External DTDs

```
<!DOCTYPE XML_doc_root_name SYSTEM  
    "DTD_file_name">
```

→ SHOW [planes.xml](#)

8.5 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 that 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 <==> { URL, URN}
 - 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:**
 1. Shorthand
 2. URI includes characters that are illegal in XML

8.5 Namespaces (continued)

- Can declare two namespaces on one element

```
<gmcars xmlns:gm = "http://www.gm.com/names"
xmlns:html =
    "http://www.w3.org/TR/xhtml1/strict">
```

- The `gmcars` element can now use `gm` names and `html` names
- One namespace can be made the default by leaving the prefix out of the declaration

8.6 XML Schemas

- **Problems with DTDs:**

1. Syntax is different from XML - cannot be parsed with an XML parser
2. It is confusing to deal with two different syntactic forms
3. DTDs do not allow specification of particular kinds of data

8.6 XML Schemas (continued)

- XML Schemas is one of the alternatives to DTD

- **Two purposes:**

1. Specify the structure of its instance XML documents
2. 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"
```

8.6 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">
```

8.6 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)

8.6 XML Schemas (continued)

- XMLS defines over **40 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

- DTDs define global elements (context is irrelevant)

- With XMLS, context is essential, and 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`

8.6 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

8.6 XML Schemas (continued)

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

- Categories of Complex Types

1. Element-only elements
2. Text-only elements
3. Mixed-content elements
4. Empty elements

- Element-only elements

- 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

8.6 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 [planes.xml](#)

- 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>
```

8.6 XML Schemas (continued)

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

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

- *Validating Instances of XML Schemas*

- Can be done with several different tools

- One of them 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 can find the schema

8.7 Displaying Raw XML Documents

- There is no presentation information in an XML document

- 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 Figure 8.2 and 8.3

8.8 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 Figure 8.4

8.9 XSLT Style Sheets

- XSL began as a standard for presentations of XML documents
 - Split into two parts:
 - **XSLT** - Transformations
 - **XSL-FO** - Formatting objects
 - XSLT uses style sheets to specify transformations

8.8 XSLT Style Sheets (continued)

- An XSLT processor merges an XML document into an XSLT style sheet
 - This merging is a template-driven process
- 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 used to connect an XSLT style sheet to an XML document

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

- An example:

```
<?xml version = "1.0" ?>
<!-- xslplane.xml -->
<?xml-stylesheet type = "text/xsl"
                href = "xslplane.xsl" ?>

<plane>
  <year> 1977 </year>
  <make> Cessna </make>
  <model> Skyhawk </model>
  <color> Light blue and white </color>
</plane>
```

8.8 XSLT Style Sheets (continued)

- 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">
```

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

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

- A template can match any element, just by naming it (in place of /)
- XSLT elements 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>
```

8.8 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 XSLT document

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

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

--> SHOW `xslplane.xml` and Figure 8.5

- The XSLT `for-each` element

- Used when an XML document has a sequence of the same elements

--> SHOW `xslplanes.xml`

→ SHOW `xslplanes.xml` & Figure 8.6

→ Look at `zooinventory` example