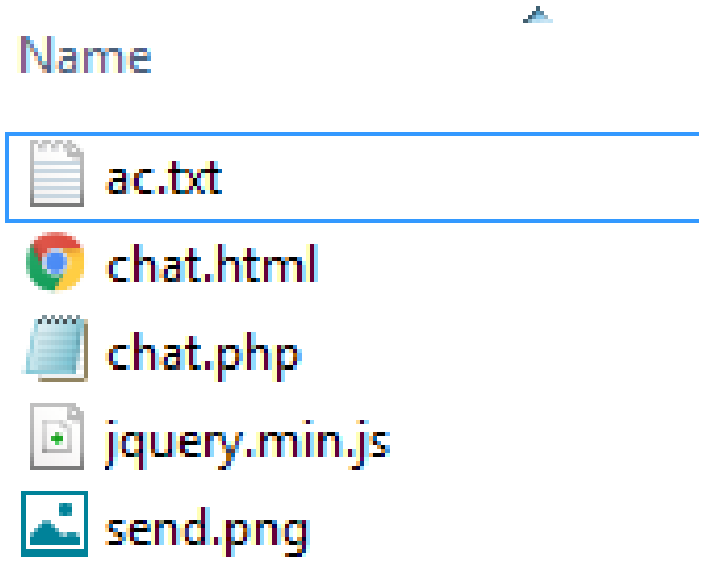
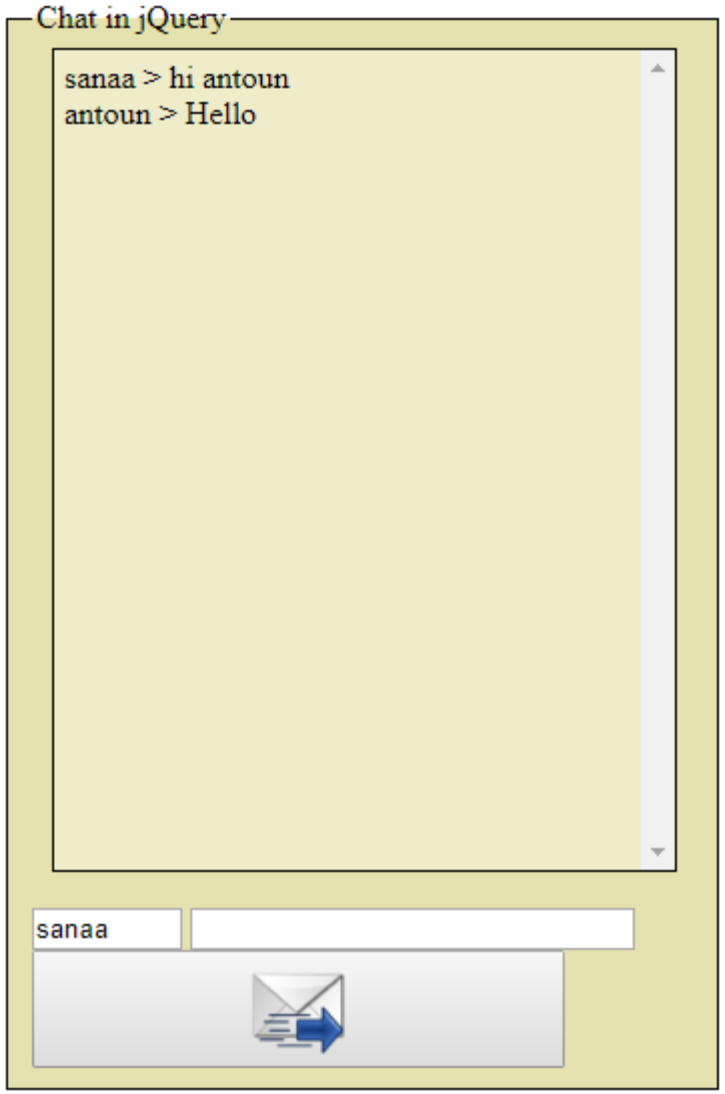
AJAX and jQuery

Chat

- This type of program allows you to enter short text messages that will be displayed in everyone who follows the conversation.





Chat

- When a user sends a message, the data form must be sent to the chat.php. This program updates the file ac.txt that you will used to refresh the contents of the conversation area.

Chat.php

```php
<?php
        $name = $_POST['name'];                          //We get the nickname and store it in a variable
        $message = $_POST['message'];                    //We do the same with the message
        $line = $name.' > '.$message."<br>";             //The message is created
        $file = file('ac.txt');                          //Read the ac.htm file and store the answer in a variable of type array
        array_unshift($file, $line);                     //Add the text to the beginning of the array
        file_put_contents('ac.txt', $file);              //We write the contents of the array in the ac.htm file
?>
```

Chat.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Chat jQuery</title>
      <style type="text/css">
      #conversation {
        width: 300px;
        height: 400px;
        border: black 1px solid;
        background-color: #efecca;
        overflow-x: hidden;
        overflow-y: scroll;
        padding: 5px;
        margin-left: 10px;
      }
      fieldset {
        width: 330px;
        background-color: #e6e2af;
        border: black 1px solid;
      }
    </style>
  </head>
```

```
<body>
    <fieldset>
     <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
     <script>
         $(function() {
             displayConversation();
             $('#send').click(function() {
                 var name = $('#name').val();
                 var message = $('#message').val();
                 $.post('chat.php', {'name':name, 'message': message }, function() {
                     displayConversation();
                 });
             });

             function displayConversation() {
             $('#conversation').load('ac.txt');
             $('#message').val('');
             $('#message').focus();
             }
             setInterval(displayConversation, 4000);
         });
     </script>

     <legend>Chat in jQuery</legend>
     <div id="conversation"></div><br />
         <form action="#" method="post">
             <input type="text" id="name" value="username" size="6">
             <input type="text" id="message" size="27">
             <button type="button" id="send" title="Send"><img src="send.png" width="20%" height="20%"></button>
         </form>
     </fieldset>
  </body>
</html>
```

ajax : post data to php page
and do something

ajax : load here
the content of a page

XML

Introduction

- HTML was developed 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

Introduction

- 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”)

- 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

Introduction

- 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*

Example Planes.xml

```
<!--
    planes.xml - A document that lists ads for  used airplanes
-->
<planes_for_sale>
  <ad>
    <year>1977</year>
    <make>Cessna</make>
    <model>Skyhawk</model>
    <color>Light blue and white</color>
    <description>
      New paint, nearly new interior, 685 hours SMOH, full IFR King avionics
    </description>
    <price>23,495</price>
    <seller phone="555-222-3333">Skyway Aircraft</seller>
    <location>
      <city>Rapid City,</city>
      <state>South Dakota</state>
    </location>
  </ad>
  <ad>
    <year>1965</year>
    <make>Piper</make>
    <model>Cherokee</model>
    <color>Gold</color>
    <description>
      240 hours SMOH, dual NAVCOMs, DME, new Cleveland brakes, great shape
    </description>
    <seller phone="555-333-2222" email="jseller@www.axl.com">John Seller</seller>
    <location>
      <city>St. Joseph,</city>
      <state>Missouri</state>
    </location>
  </ad>
</planes_for_sale>
```

Example Planes.dtd

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.dtd - a document type definition for
                  the planes.xml document, which specifies
                  a list of used airplanes for sale  -->

<!ELEMENT planes_for_sale (ad+)>
<!ELEMENT ad (year, make, model, color, description,
              price?, seller, location)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT seller (#PCDATA)>
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>

<!ATTLIST seller phone CDATA #REQUIRED>
<!ATTLIST seller email CDATA #IMPLIED>

<!ENTITY c "Cessna">
<!ENTITY p "Piper">
<!ENTITY b "Beechcraft">
```

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. For a particular XML tag set, either a document type definition (DTD) or an XML schema

The Syntax of XML

- *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)

The Syntax of XML

- All XML documents begin with an XML declaration:

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

- X*ML 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)

The Syntax of XML

- *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>
```

The Syntax of XML

- 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)

The Syntax of XML

```
<!-- 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>
```

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

XML Document Structure

- *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!)

XML Document Structure

- 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;`

XML Document Structure

- Predefined entities (as in XHTML):

<	<
>	>
&	&
"	"
'	'

XML Document Structure

- 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

Document 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 and define a markup language for them
- The DTD for a document can be internal or external
- 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`

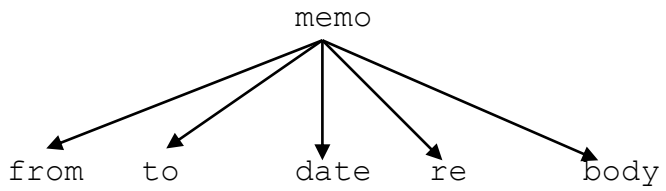
Document Type Definitions *Declaring Elements*

- An element declaration specifies the name 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)

Document Type Definitions
Declaring Elements

- General form:
  `<!ELEMENT` element_name `(`list of child names`)>`
  e.g.,

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



Document Type Definitions *Declaring Elements*

- 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 *p*arsable *c*haracter *data*
  - Data type could also be `EMPTY` (no content) and `ANY` (can have any content)
  - Example of a leaf declaration:
    ```
    <! LEMENT name (#PCDATA)>
    ```

Document Type Definitions *Declaring Attributes*

- General form:
  `<!ATTLIST` el_name at_name at_type [default]>
- *Attribute types*: there are ten different types, 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)

```
<!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>
```

Document Type Definitions
Declaring Entities

- *Two kinds*:

1. A *general entity* can be referenced anywhere in the content of an XML document

2. 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"`>`

Example Planes.xml

```
<!--
    planes.xml - A document that lists ads for  used airplanes
-->
<planes_for_sale>
  <ad>
    <year>1977</year>
    <make>Cessna</make>
    <model>Skyhawk</model>
    <color>Light blue and white</color>
    <description>
      New paint, nearly new interior, 685 hours SMOH, full IFR King avionics
    </description>
    <price>23,495</price>
    <seller phone="555-222-3333">Skyway Aircraft</seller>
    <location>
      <city>Rapid City,</city>
      <state>South Dakota</state>
    </location>
  </ad>
  <ad>
    <year>1965</year>
    <make>Piper</make>
    <model>Cherokee</model>
    <color>Gold</color>
    <description>
      240 hours SMOH, dual NAVCOMs, DME, new Cleveland brakes, great shape
    </description>
    <seller phone="555-333-2222" email="jseller@www.axl.com">John Seller</seller>
    <location>
      <city>St. Joseph,</city>
      <state>Missouri</state>
    </location>
  </ad>
</planes_for_sale>
```

Example Planes.dtd

```
<?xml version = "1.0" encoding = "utf-8"?>
<!-- planes.dtd - a document type definition for
                  the planes.xml document, which specifies
                  a list of used airplanes for sale  -->

<!ELEMENT planes_for_sale (ad+)>
<!ELEMENT ad (year, make, model, color, description,
              price?, seller, location)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT make (#PCDATA)>
<!ELEMENT model (#PCDATA)>
<!ELEMENT color (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT seller (#PCDATA)>
<!ELEMENT location (city, state)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>

<!ATTLIST seller phone CDATA #REQUIRED>
<!ATTLIST seller email CDATA #IMPLIED>

<!ENTITY c "Cessna">
<!ENTITY p "Piper">
<!ENTITY b "Beechcraft">
```

Document Type Definitions
XML Parsers

- Always check for well formedness
- Some check for validity, relative to a given DTD
  - Called *validating XML parsers*
- You can download a validating XML parser from:
  `http://xml.apache.org/xerces-j/index.html`

Document Type Definitions

- Internal DTDs

```
<!DOCTYPE root_name [
    …
]>
```

- External DTDs

<!DOCTYPE *XML_doc_root_name* SYSTEM
"DTD_file_name">

Recent browsers will mostly not load external DTDs

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!

Namespaces

- 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
  - 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

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

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

XML Schemas

- 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

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
- Every XML schema itself defines a tag set, which must be named
  `targetNamespace = "http://cs.uccs.edu/planeSchema"`

XML Schemas

- 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"`

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“
>
```

XML Schemas
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/planeSchemaplanes.xsd" >
```

Planes.xsd

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

<!-- planes.xsd
     A simple schema for planes.xml
     -->

<xsd:schema
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    targetNamespace = "http://cs.uccs.edu/planeSchema"
    xmlns = "http://cs.uccs.edu/planeSchema"
    elementFormDefault = "qualified">

    <xsd:element name = "planes">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name = "make"
                             type = "xsd:string"
                             minOccurs = "1"
                             maxOccurs = "unbounded" />
            </xsd:all>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

XML Schemas
Data Type Categories

1. Simple (strings only, no attributes and no nested elements)
2. Complex (can have attributes and nested elements)

- 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

XML Schemas
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"
```

XML Schemas
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

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

XML Schemas
User-Defined Types

- 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

```
<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>
```

XML Schemas
User-Defined Types

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

Planes.xsd

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

<!-- planes.xsd
     A simple schema for planes.xml
     -->

<xsd:schema
    xmlns:xsd = "http://www.w3.org/2001/XMLSchema"
    targetNamespace = "http://cs.uccs.edu/planeSchema"
    xmlns = "http://cs.uccs.edu/planeSchema"
    elementFormDefault = "qualified">

    <xsd:element name = "planes">
        <xsd:complexType>
            <xsd:all>
                <xsd:element name = "make"
                             type = "xsd:string"
                             minOccurs = "1"
                             maxOccurs = "unbounded" />
            </xsd:all>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Planes.xml

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

<!-- planes.xsd.xml
     A simple XML document for illustrating a schema
     The schema is in planes.xsd
     -->
<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">
    <make> Cessna </make>
    <make> Piper </make>
    <make> Beechcraft </make>
</planes>
```

XML Schemas
User-Defined 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>
```

XML Schemas
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

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 Planes.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- planes.xml - A document that lists ads for used airplanes -->
<!DOCTYPE planes_for_sale SYSTEM "planes.dtd">
- <planes_for_sale>
  - <ad>
      <year> 1977 </year>
      <make> </make>
      <model> Skyhawk </model>
      <color> Light blue and white </color>
      <description> New paint, nearly new interior, 685 hours SMOH, full IFR King avionics </description>
      <price> 23,495 </price>
      <seller phone="555-222-3333"> Skyway Aircraft </seller>
    - <location>
        <city> Rapid City, </city>
        <state> South Dakota </state>
      </location>
    </ad>
  - <ad>
      <year> 1965 </year>
      <make> </make>
      <model> Cherokee </model>
      <color> Gold </color>
      <description> 240 hours SMOH, dual NAVCOMs, DME, new Cleveland brakes, great shape </description>
      <seller phone="555-333-2222" email="jseller@www.axl.com"> John Seller </seller>
    - <location>
        <city> St. Joseph, </city>
        <state> Missouri </state>
      </location>
    </ad>
</planes_for_sale>
```

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"?>
```

1977 Cessna Skyhawk

Light blue and white

New paint, nearly new interior, 685 hours SMOH, full IFR King avionics

23,495

Skyway Aircraft

Rapid City, South Dakota

1965 Piper Cherokee

Gold

240 hours SMOH, dual NAVCOMs, DME, new Cleveland brakes, great shape

John Seller

St. Joseph, Missouri

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 DTDs or schemas) into the document
  4. 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:*
  1. A browser with an XML parser
  2. A stand-alone XML parser

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

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

- *Web Service Definition Language* (WSDL): Used to describe available services, as well as of 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