CSC 498: Web Programming

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

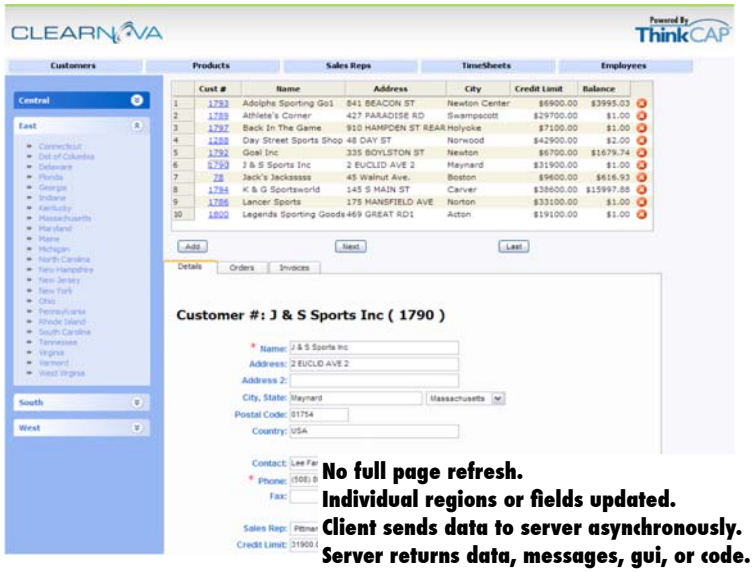
AJAX Defined

- Asynchronous Javascript and XML
  - Term coined by Jesse James Garrett
  - www.adaptivepath.com
- AJAX is not a new technology
  - Google calls their technique: Javascript
  - Also known as XMLHTTP technique
  - In use since at least 1997
- A bundle of techniques
  - XML data interchange only
  - Passing Javascript methods to client
  - DHTML widgets
  - XML & XSLTs
- Core techniques are centered around asynchronous communication to the server without a page refresh

So why is AJAX so hot—NOW?

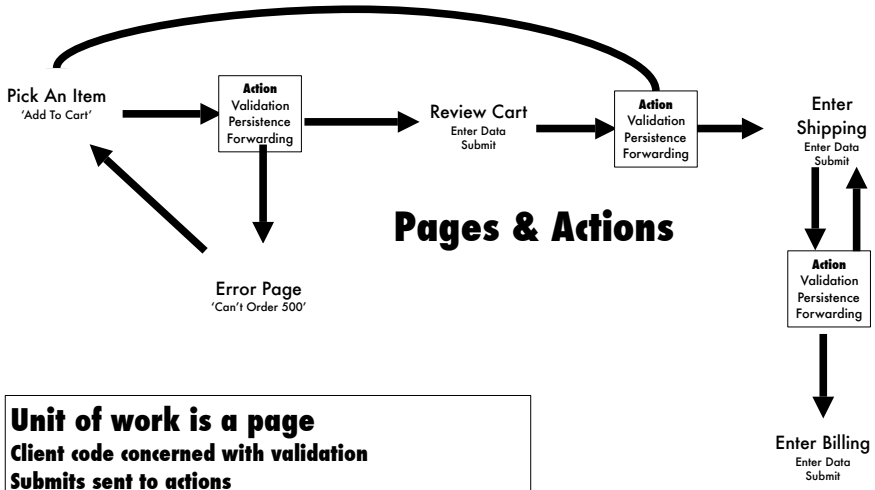
- Demand for richer applications is growing
  - Broadband means we can—and want to—do more
- It has a name
  - Think LAMP—helped solidify the thoughts/techniques in people's minds
- Recent Google applications have sparked people's imagination
  - Google gmail, Google suggests, Google Maps
- People are thinking of building APPLICATIONS…not just sites
- The Tipping Point
  - All of this has made rich internet apps reach its tipping point—where adoption spreads rapidly and dramatically

AJAX Enabled Master Detail Form



No full page refresh.
Individual regions or fields updated.
Client sends data to server asynchronously.
Server returns data, messages, gui, or code.

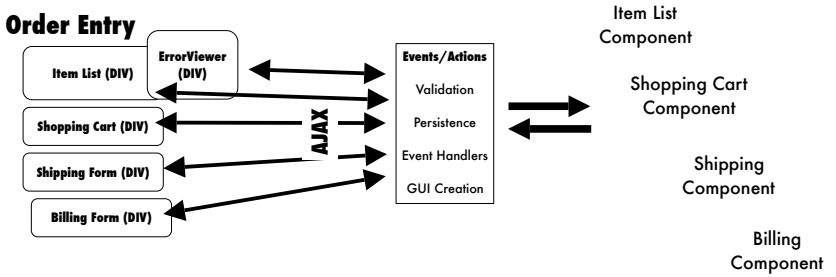
Traditional Web Applications...



Unit of work is a page
Client code concerned with validation
Submits sent to actions
Actions perform work and then forward to next page
Page refresh for each submit

AJAX Changes How Web Apps are Built

Components & Events



Unit of work is a component
Three-Tier Client/Server Model
Client code has validation, flow, layout, data interchange
No submit buttons—save buttons
Only parts of pages are updated at a time

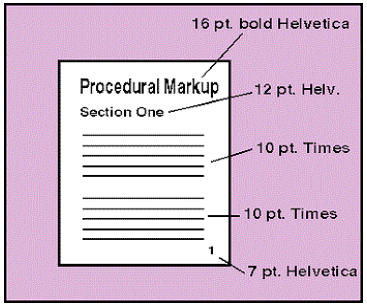
XML?

XML

- XML is a markup language
- Markup languages originate from the book publishing industry.
  - Allows us to embed formatting instructions in the document.
- Two types of markup:
  - Procedural markup
  - Descriptive markup or Generic markup

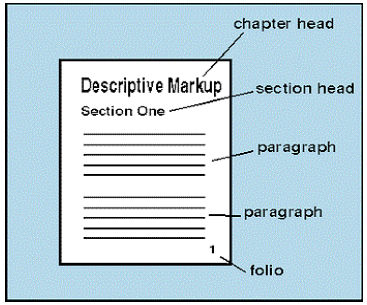
Procedural Markup

- Formatting codes are mixed in with the text of the document.
- Typically unique to a specific software package such as Microsoft Word
- Disadvantages:
  - a single way of presenting the information, such as a printed page
  - provide no capability to define appearance for other media, such as CD-ROM and Internet.



Descriptive Markup

- Describes the purpose of the text in a document, rather than its physical appearance on the page.
  - content of a document remains separate from its style.
  - Based on the structure of a document and identifies elements within that structure -- such as a chapter, a section, or a table of contents -- using notations that describe what the element is, not how it appears.
- Allows multiple presentations of the same information.
  - e.g. publications on paper, on CD-ROM and on the Web can come from one set of source files.



SGML

- SGML: Standard Generalized Markup Language
  - an international standard (ISO 8879) published in 1986.
  - prescribes a standard format for embedding descriptive markup within a document.
  - specifies a standard method for describing the structure of a document.
    - allows you to set up hierarchical models for each type of document you produce.
    - supports different document structures for different information types, e.g. technical manuals, parts catalogs, design specifications, reports and memos.

SGML

- SGML documents are
  - ASCII format: human and machine-readable
  - independent of any specific hardware or software
  - portable
- Can be broken into three components:
  - structure
  - content
  - style

SGML

- Structure:
  - defined in a file called the *Document Type Definition* (DTD)
  - describes the structure of a document, much like a database schema describes the types of information it handles and the relationships between fields
    - e.g. the address information should contain street, city, country fields
  - specifies rules for the relationships between elements
    - e.g. the street field must go first before the city field.
    - e.g. the country field is compulsory and can not be blanks.

SGML

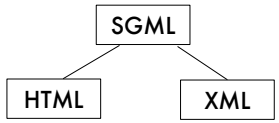
```
<address>
        <street> street name </street>
        <city> city name </city>
</address>
```

- Content:
  - content is the information itself
  - identify the content's position within the DTD structure by using "tagging."
    - These tags mark the beginning and end of each part of the structure.
    - e.g. <street> 276 Castle Peak Road </street>
    - "<street>" indicates the start of a street name, and "</street>" indicates the end
  - Nesting of tags reveal the structure of a document.

SGML

- Style:
  - Define how different fields should be displayed.
    - e.g. The style defines that "city field in the address should be all in upper case and in red colored font".
    - Content: <city> hong kong </city>
    - Display: HONG KONG
  - SGML itself has nothing to do with setting standards for style, so most systems still rely on proprietary methods.

SGML



- Two implementations of SGML standard:
  - HTML: Hypertext Markup Language
  - XML: Extensible Markup Language

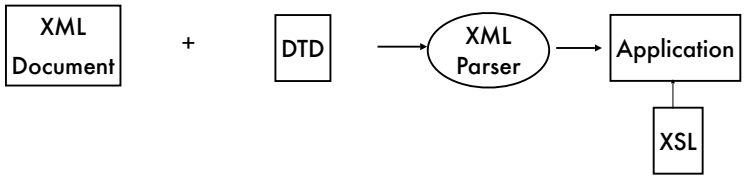
	HTML	XML
Structure	Not defined	Defined in DTD files
Content	▪ Limited number of tags ▪ Tags are defined in HTML specifications ▪ Tags are not case sensitive	▪ Unlimited number of tags ▪ Tags are defined in DTD ▪ Tags are case sensitive
Style	Not defined	Defined in XSL (style) files

Markup Languages

Language	Year Introduced	Description
SGML	1986	International standard for specifying a document markup language
HTML	1992	An early recommendation
HTML+	1993	A superset of HTML
HTML 2.0	1995	An HTML version created by IETF
HTML 3.2	1996	W3C formal recommendation for HTML
XML	1998	Universal format for structured documents and data on the Web
XHTML	2000	Document markup language for the Web

XML

- XML: Extensible Markup Language
  - was developed in 1996 by the World Wide Web Consortium (W3C) XML working group
  - Designed to carry data
  - Has all 3 components of SGML:
    - structure (DTD)
    - content (XML)
    - style (XSL)



XML Example

```
<bank>
    <account>
        <account-number> A-101    </account-number>
        <branch-name>    Downtown </branch-name>
        <balance>    500         </balance>
    </account>
    <depositor>
        <account-number> A-101    </account-number>
        <customer-name> Johnson </customer-name>
    </depositor>
</bank>
```

start tag

end tag

- human and machine readable
- self documenting
- based on nested tags
- Unlike in HTML, you can define your own tags

XML

- XML documents contain only data
  - Applications decide how to display the data
  - Files end in the .xml extension
  - Highly portable
  - Language for creating markup languages
    - Can create new tags
- Document Type Definition (DTD) files
  - Defines grammatical rules for the document
  - Used to check the XML document structure against
- Extensible Style Language (XSL) files
  - Defines additional information for rendering the document
  - Possible to search, sort, manipulate and render XML using Extensible Style Language (XSL)

XML

- XML parser
  - Checks an XML document's syntax
    - validating against DTD
    - non-validating against DTD
  - Support either the
    - Document Object Model (DOM)
      - Build a tree structure containing the XML document's data
    - Simple API for XML (SAX)
      - Process the document and generate events
  - More info on XML parsers:
    - www.xml.com/xml/pub/Guide/XML_Parsers

Basic XML Documents

- XML elements
  - Root element
    - Must be exactly one per XML document
    - Contains all other elements in document
    - Lines preceding the root element are called the prolog
  - Container element
    - Contains sub-elements (children)
  - Empty element
    - No matching end tag
    - In HTML, IMG
    - Terminate with forward slash (/)

Well-formed Documents

- A document which conforms to the XML syntax is called well-formed.
  - all tag properly nested
  - single top level element
  - ...
- How to check this?
  - load the XML-file in your web browser.
  - Use NetBeans, eclipse, …
  - many more tools ...

Sample XML Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

- XML documents use a self-describing and simple syntax
  - The first line is the XML declaration.
    - Defines the XML version (1.0) and the encoding used
    - The next line (<note>)describes the root element of the document
      - "this document is a note"
    - The next 4 lines describe 4 child elements of the root (to, from, heading, and body)
    - The last line defines the end of the root element, </note>

Structure of XML Data

- **Tag**: label in angle brackets for a section of data
- **Element**: section of data beginning with start tag <*tagname*> and ending with matching end tag </*tagname*>

```
<tagname>
    <!-- comment, place the content of your
         element here -->
</tagname>
```

- Can invent our own tags.

```
<ca-super-tag>  </ca-super-tag>
```

- Every document must have a single top-level element

Structure of XML Data

- Elements must be properly nested

```
<bank>
  <customer>
    <customer-name>   Hayes  </customer-name>
    <customer-street> Main  </customer-street>
    <customer-city>   Harrison  </customer-city>
    <account>
      <account-number> A-102 </account-number>
      <branch-name>    Perryridge </branch-name>
      <balance>        400 </balance>
    </account>
    <account>
      <account-number> A-205 </account-number>
      <branch-name>    Oxford street </branch-name>
      <balance>        329 </balance>
    </account>

  </customer>
```


Attributes

- Elements can have attributes

```
<account acct-type = "checking" >
    <account-number> A-102   </account-number>
    <branch-name> Perryridge </branch-name>
    <balance>      400       </balance>
</account>
```

  Attributes are specified by *name*=*value* pairs inside the starting tag of an element

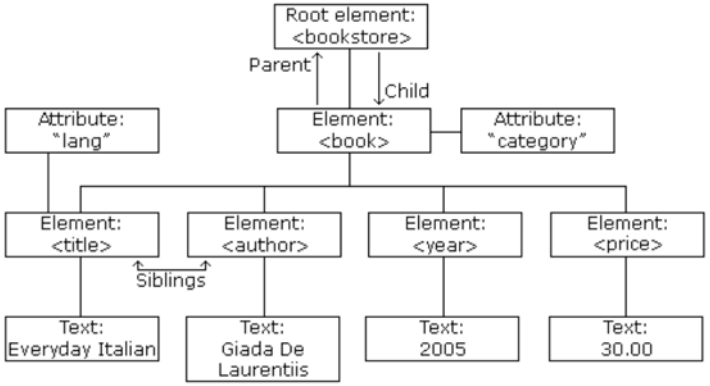
- An element may have several attributes, but each attribute name can only occur once

```
<account  acct-type = "checking"  monthly-fee= "5">
```


Another Example

```
<bookstore>
<book category="COOKING">
  <title lang="en">Everyday Italian</title>
  <author>Giada De Laurentiis</author>
  <year>2005</year>
  <price>30.00</price>
</book>
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
```


Example



Document Type Definition (DTD)

- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax

```
<!ELEMENT element (subelements-specification) >
<!ATTLIST   element (attributes)  >
```


Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Example

```
<! ELEMENT depositor (customer-name  account-number)>
<! ELEMENT customer-name (#PCDATA)>
<! ELEMENT account-number (#PCDATA)>
```

- Subelement specification may have regular expressions

```
<!ELEMENT bank ( ( account | customer | depositor)+)>
```

  - Notation:
    - "|" - alternatives
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences

Bank DTD

```
<!ELEMENT bank ( ( account | customer | depositor)+)>
<!ELEMENT account (branch-name, balance)>
<!ELEMENT branch-name (#PCDATA)>
<!ELEMENT balance (#PCDATA)>
<!ELEMENT customer
        (customer-name, customer-street, customer-city)>
<!ELEMENT customer-name (#PCDATA)>
<!ELEMENT customer-street (#PCDATA)>
<!ELEMENT customer-city (#PCDATA)>
<!ELEMENT depositor (customer-name, account-number)>
```

Notation:
"|" - alternatives
"+" - 1 or more occurrences
"*" - 0 or more occurrences

Document Type Definitions

- DTD: Document Type Definitions
  - Specify list of element types, attributes and their relationships to each other
  - Optional, but recommended for program conformity
  - !Element
    - Element type declaration – defines the rules for an element
    - #PCDATA: The element can store parsed character data
  - !ATTLIST
    - Defines attributes for an element
    - #IMPLIED: Can assign its own type attribute or ignore
    - #REQUIRED: The specified attribute must be declared in the document
    - #FIXED: The Specified attribute must be declared with given value

Namespaces

- In XML, element names are defined by the developer
  - Results in a conflict when trying to mix XML documents from different XML applications
- Example
  - Cannot add the following two tables together due to name conflict

```
<table>
   <tr>
   <td>Apples</td>
   <td>Bananas</td>
   </tr>
</table>
```

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


Namespaces

- Name conflicts in XML can easily be avoided using a name prefix

```
<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>
```


Namespaces

- When using prefixes in XML, a so-called namespace for the prefix must be defined.
  - The namespace is defined by the *`xmlns`* attribute in the start tag of an element.
  - The namespace declaration has the following syntax.
    `xmlns:prefix="URI"`

Namespaces

- In the shown example, the xmlns attribute in the <table> tag give the h: and f: prefixes a qualified namespace.
- When a *namespace* is defined for an element, all child elements with the same prefix are associated with the same namespace.

```
<root>
<h:table xmlns:h="http://www.w3.org/TR/
  html4/">
  <h:tr>
  <h:td>Apples</h:td>
  <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://
  www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```


Namespaces

- XML is often used for data exchange between organizations.
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents.
- Need unique names for elements.
- Better solution: use unique-name:element-name.
- unique-name is called namespace.

Namespaces

- Avoid using long unique names all over document by using defining a short namespace prefix within a document.
- FB:branch is just an abbreviation to make the file more readable.

```
<bank Xmlns:FB='http://www.FirstBank.com'>
        …
   <FB:branch>
              <FB:branchname> Downtown </FB:branchname>
        <FB:branchcity> Brooklyn </FB:branchcity>
   </FB:branch>
  …
</bank>
```

- The full (qualified) name of the branch element is http://www.firstbank.com:branch
- More on namespaces can be found at: http://www.w3schools.com/xml/xml_namespaces.asp

Examples

```
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />
```

•"square" element is defined to be an empty element with a "width" attribute of type CDATA.
•If no width is specified, it has a default value of 0

```
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />
```

•Use the #IMPLIED keyword if you don't want to force the author to include an attribute, and you don't have an option for a default value.

Examples

DTD:
<!ATTLIST person number CDATA #REQUIRED>

Valid XML:
<person number="5677" />

Invalid XML:
<person />

Document Type Definitions

- <?xml version="1.0" ?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>

s

<?xml version="1.0" ?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>

Note.xml with DTD included

```
<?xml version="1.0"?>
<!DOCTYPE note [
   <!ELEMENT note (to, from, heading, body)>
   <!ELEMENT to (#PCDATA)>
   <!ELEMENT from (#PCDATA)>
   <!ELEMENT heading (#PCDATA)>
   <!ELEMENT body (#PCDATA)>
]>
<note>
    <to>      Tove    </to>
    <from>    Jani     </from>
    <heading> Reminder </heading>
    <body>Don't forget me this weekend!</body>
</note>
```

DTD

Attribute Specification in DTD

- Attribute specification : for each attribute
  - Name
  - Type of attribute
    - CDATA
    - ID (identifier) or IDREF (ID reference) or IDREFS (multiple IDREFs)
      - more on this later
  - Whether
    - mandatory (#REQUIRED)
    - has a default value (value),
    - or neither (#IMPLIED)
- Examples
  - <!ATTLIST account acct-type CDATA "checking">
  - <!ATTLIST customer
    customer-id ID # REQUIRED
    accounts IDREFS # REQUIRED >

IDs and IDREFs

- IDREF
  - The value is the id of another element
- IDREFS
  - The value is a list of other ids
- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  - Thus the ID attribute value is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values.
- Each ID value must contain the ID value of an element in the same document

Bank DTD with ID and IDREF Attributes

```
<!ELEMENT bank2 ( ( account | customer )+)>
<!ELEMENT account (branch, balance)>
<!ATTLIST account
    account-number ID          #REQUIRED
    owners         IDREFS      #REQUIRED>
<!ELEMENT customer
      (customer-name, customer-street, customer-city)>
<!ATTLIST customer
      customer-id     ID          #REQUIRED
      accounts        IDREFS      #REQUIRED>
<!ELEMENT branch (#PCDATA)>
<!ELEMENT balance (#PCDATA)>
<!ELEMENT customer-name (#PCDATA)>
<!ELEMENT customer-street (#PCDATA)>
<!ELEMENT customer-city (#PCDATA)>
```


XML data with ID and IDREF attributes

```
<bank2>
   <account account-number="A-401" owners="C100 C102">
      <branch> Downtown </branch>
      <balance>        500 </balance>
   </account>
   <account account-number="A-402" owners="C102">
      <branch> Downtown </branch>
      <balance>  8000 </balance>
   </account>
   <customer customer-id="C100" accounts="A-401">
      <customer-name>   Joe        </customer-name>
      <customer-street>  Monroe   </customer-street>
      <customer-city>     Madison </customer-city>
   </customer>
   <customer customer-id="C102" accounts="A-401 A-402">
      <customer-name>   Mary     </customer-name>
      <customer-street>   Erin      </customer-street>
      <customer-city>      Newark </customer-city>
   </customer>
</bank2>
```


Note.xml with a reference to DTD

note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

note.xml

Reference to external file

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "c:\comp302\note.dtd">
<note>
    <to>      Tove    </to>
    <from>    Jani     </from>
    <heading> Reminder </heading>
    <body>Don't forget me this weekend!</body>
</note>
```


Note.xml with a reference to DTD

note.dtd

```
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

note.xml

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "http://www.csc.liv.ac.uk/
~christoph/note.dtd">
<note>
    <to>      Tove    </to>
    <from>    Jani     </from>
    <heading> Reminder </heading>
    <body>Don't forget me this weekend!</body>
</note>
```

refernce to DTD at URL

Limitations of DTDs

- No typing of text elements and attributes
  - All values are strings, no integers, reals, etc.
- Difficult to specify unordered sets of subelements
  - Order is usually irrelevant in databases
  - (A | B)* allows specification of an unordered set, but
    - Cannot ensure that each of A and B occurs only once
- IDs and IDREFs are untyped
  - The *owners* attribute of an account may contain a reference to another account, which is meaningless
    - *owners* attribute should ideally be constrained to refer to customer elements

XML Document Schema

- When we build a database, we first design the database schema.
- Database schemas constrain what information can be stored in the database.
  - names of tables and attributes
  - data types of attributes,
  - the order they appear in the table
- By requiring that the data conforms to the schema, we give data a meaning.
- XML documents are not required to have a schema.
- In practice, they are very important.
  - e.g. when defining a new file format based on XML
  - or for data exchange

XML Schema Mechanisms

- Document Type Definition (DTD)
  - Widely used
- XML Schema
  - Newer, increasing use
- A document that confirms with its DTD or XML schema is called valid.
- There exist tools to check whether a XML-file is valid.
  - www.xmlvalidation.com
  - NetBeans
  - many others
- A validator checks also whether the XML-file is well-formed.

XML Schema

- XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs.
- Supports
  - Typing of values
    - E.g. integer, string, etc
    - Also, constraints on min/max values
  - User defined types
  - Is itself specified in XML syntax, unlike DTDs
    - More standard representation, but verbose
  - Is integrated with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..
- BUT: significantly more complicated than DTDs, not yet widely used.

Note.xsd (xml schema definition)

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.liv.ac.uk">
        <xs:element name="note">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="to" type="xs:string"/>
                <xs:element name="from" type="xs:string"/>
                <xs:element name="heading" type="xs:string"/>
                <xs:element name="body" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```


Note.xml

```
<?xml version="1.0"?>
<note    xmlns="http://www.liv.ac.uk"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.liv.ac.uk note.xsd">

    <to>      Tove</to>
    <from>    Jani</from>
    <heading> Reminder </heading>
    <body>Don't forget me this weekend!</body>
</note>
```

XML schema for http://www.liv.ac.uk namespace (same path as xml file)

Schema could be stored on the web:

```
xsi:schemaLocation=
"http://www.liv.ac.uk http://www.liv.ac.uk/~christoph/note.xsd"
```


Javascript?

Javascript

- client-side programming language
  - programs are written in a separate programming language
    - e.g., JavaScript, JScript, VBScript
  - programs are embedded in the HTML of a Web page, with tags to identify the program component
    - e.g., <script type="text/javascript"> … </script>
  - the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML

Scripts vs. Programs

- A scripting language is a simple, interpreted programming language
  - scripts are embedded as plain text, interpreted by application
  - simpler execution model: don't need compiler or development environment
  - saves bandwidth: source code is downloaded, not compiled executable
  - platform-independence: code interpreted by any script-enabled browser
  - but: slower than compiled code, not as powerful/full-featured

Scripts vs. Programs

- JavaScript: the first Web scripting language, developed by Netscape in 1995
  - syntactic similarities to Java/C++, but simpler, more flexible in some respects,
  - limited in others
    - loose typing, dynamic variables, simple objects, …
  - JScript: Microsoft version of JavaScript, introduced in 1996
    - same core language, but some browser-specific differences
    - fortunately, IE, Netscape, and Firefox can (mostly) handle both
    - JavaScript & JScript
    - JavaScript 1.5 & JScript 5.0 cores conform to ECMAScript standard
  - VBScript: client-side scripting version of Microsoft Visual Basic

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data (probably the most commonly used application)
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies
- defining programs with Web interfaces
  - utilize buttons, text boxes, clickable images, prompts, etc

limitations of client-side scripting

- since script code is embedded in the page, it is viewable to the world
- for security reasons, scripts are limited in what they can do
  - e.g., can't access the client's hard drive
- since designed to run on any machine platform, scripts do not contain platform specific commands
- script languages are not full-featured
  - e.g., JavaScript objects are crude, not good for large project development

JavaScript

- JavaScript code can be embedded in a Web page using SCRIPT tags
  - the output of JavaScript code is displayed as if directly entered in HTML

```
<html>
<!-- COMP519  js01.html  16.08.06 -->

<head>
  <title>JavaScript Page</title>
</head>

<body>
  <script type="text/javascript">
    // silly code to demonstrate output

    document.write("Hello world!");

    document.write("<p>How are <br/>" +
                   "<i>you</i>?</p>");
  </script>

  <p>Here is some static text as well.
  </p>
</body>
</html>
```

view page

`document.write` displays text in page

text to be displayed can include HTML tags

the tags are interpreted by the browser when the text is displayed

as in C++/Java, statements end with `;`
but a line break is also interpreted as the end of a statement

JavaScript comments similar to C++/Java

`//` starts a single line comment

`/*...*/` enclose multi-line comments

JavaScript Data Types & Variables

- JavaScript has only three primitive data types

  String : `"foo"  'howdy do'  "I said 'hi'."  ""`
  Number: `12  3.14159  1.5E6`
  Boolean : `true  false`

```
<html>
<!-- COMP519  js02.html  16.08.06 -->

<head>
  <title>Data Types and Variables</title>
</head>

<body>
  <script type="text/javascript">
    var x, y;
    x= 1024;

    y=x;  x = "foobar";
    document.write("<p>x = " + y + "</p>");
    document.write("<p>x = " + x + "</p>");
  </script>
</body>
</html>
```

view page

assignments are as in C++/Java

```
message = "howdy";
pi = 3.14159;
```

variable names are sequences of letters, digits, and underscores that *start with a letter or an underscore*

variables names are case-sensitive

you don't have to declare variables, will be created the first time used, but it's better if you use var statements

```
var message, pi=3.14159;
```

variables are loosely typed, can be assigned different types of values

JavaScript Operators & Control Statements

```
<html>
<!-- COMP519 js03.html  16.08.06 -->

<head>
  <title>Folding Puzzle</title>
</head>

<body>
 <script type="text/javascript">
    distanceToSun = 93.3e6*5280*12;
    thickness = .002;

    foldCount = 0;
    while (thickness < distanceToSun) {
        thickness *= 2;
        foldCount++;
    }
    document.write("Number of folds = " +
                   foldCount);
  </script>
</body>
</html>
```

view page

standard C++/Java operators & control statements are provided in JavaScript

- `+, -, *, /, %, ++, --, …`
- `==, !=, <, >, <=, >=`
- `&&, ||, !,===,!==`
- if-then, if-then-else, switch
- while, for, do-while, …

PUZZLE:
Suppose you took a piece of paper and folded it in half, then in half again, and so on.

How many folds before the thickness of the paper reaches from the earth to the sun?

**Find more info on this subject*

JavaScript Math Routines

```
<html>
<!-- COMP519  js04.html  16.08.06 -->

<head>
  <title>Random Dice Rolls</title>
</head>

<body>
  <div style="text-align:center">
    <script type="text/javascript">
      roll1 = Math.floor(Math.random()*6) + 1;
      roll2 = Math.floor(Math.random()*6) + 1;

      document.write("<img src='http://www.csc.liv.ac.uk/"+
                "~martin/teaching/comp519/Images/die" +
                roll1 + ".gif'/>");
      document.write("  ");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                "~martin/teaching/comp519/Images/die" +
                roll2 + ".gif'/>");
    </script>
  </div>
</body>
</html>
```

view page

the Math object contains functions and constants

```
Math.sqrt
Math.pow
Math.abs
Math.max
Math.min
Math.floor
Math.ceil
Math.round

Math.PI
Math.E
```

`Math.random` function returns number in [0..1)

Interactive Pages Using Prompt

```
<html>
<!-- COMP519  js05.html  16.08.2006 -->

<head>
  <title>Interactive page</title>
</head>

<body>
 <script type="text/javascript">
    userName = prompt("What is your name?", "");

    userAge = prompt("Your age?", "");
    userAge = parseFloat(userAge);

    document.write("Hello " + userName + ".")
    if (userAge < 18) {
      document.write("  Do your parents know " +
                     "you are online?");
    }
    else {
      document.write("  Welcome friend!");
    }
</script>

  <p>The rest of the page...</p>
</body>
</html>
```

view page

crude user interaction can take place using prompt

- 1st argument: the prompt message that appears in the dialog box
- 2nd argument: a default value that will appear in the box (in case the user enters nothing)
- the function returns the value entered by the user in the dialog box (a string)
- if value is a number, must use parseFloat to convert

forms will provide a better interface for interaction *(later)*

User-Defined Functions

- function definitions are similar to C++/Java, except:
  - no return type for the function (since variables are loosely typed)
  - no types for parameters (since variables are loosely typed)
  - by-value parameter passing only (parameter gets copy of argument)

```
function isPrime(n)
// Assumes: n > 0
// Returns: true if n is prime, else false
{
  if (n < 2) {
    return false;
  }
  else if (n == 2) {
    return true;
  }
  else {
      for (var i = 2; i <= Math.sqrt(n); i++) {
        if (n % i == 0) {
          return false;
        }
      }
      return true;
  }
}
```

can limit variable scope

if the first use of a variable is preceded with `var`, then that variable is local to the function

for modularity, should make all variables in a function local

User-Defined Functions

- Functions can be defined both in the <head> and in the <body> section of a document.
- In order to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

Function Example

```
<html>
<!-- COMP519  js06.html  16.08.2006 -->

<head>
  <title>Prime Tester</title>

  <script type="text/javascript">
    function isPrime(n)
    // Assumes: n > 0
    // Returns: true if n is prime
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }
  </script>
</head>

<body>
 <script type="text/javascript">
    testNum = parseFloat(prompt("Enter a positive integer", "7"));

    if (isPrime(testNum)) {
      document.write(testNum + " <b>is</b> a prime number.");
    }
    else {
      document.write(testNum + " <b>is not</b> a prime number.");
    }
  </script>
</body>
</html>
```

view page

function definitions go in the HEAD

HEAD is loaded first, so the function is defined before code in the BODY is executed

```
<html>
<!-- COMP519  js07.html  16.08.2006 -->

<head>
  <title> Random Dice Rolls Revisited</title>

  <script type="text/javascript">
    function RandomInt(low, high)
    // Assumes: low <= high
    // Returns: random integer in range [low..high]
    {
      return Math.floor(Math.random()*(high-low+1)) + low;
    }
  </script>
</head>

<body>
  <div align="center">
    <script type="text/javascript">
      roll1 = RandomInt(1, 6);
      roll2 = RandomInt(1, 6);

      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/comp519/Images/die" +
                     roll1 + ".gif'/>");
      document.write("  ");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/comp519/Images/die" +
                     roll2 + ".gif'/>");
</script>
  </div>
</body>
</html>
```

view page

Another Example

Dynamic dice page

could define a function for generating random numbers in a range, then use whenever needed

easier to remember, promotes reuse

JavaScript Libraries

- If you define functions that may be useful to many pages, store in a separate library file and load the library when needed
- A file random.js may contain definitions of the following functions:
  - RandomNum(low, high) returns random real in range [low..high)
  - RandomInt(low, high) returns random integer in range [low..high)
  - RandomChar(string) returns random character from the string
  - RandomOneOf([item1,…,itemN]) returns random item from list/array
  - Note: as with external style sheets, no tags in the JavaScript library file
- load a library using the SRC attribute in the SCRIPT tag (nothing between the tags)

```
<script type="text/javascript"
        src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js">
</script>
```


Library Example

```
<html>
<!-- COMP519  js08.html  16.08.2006 -->

<head>
  <title> Random Dice Rolls Revisited</title>

  <script type="text/javascript"
    src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js">
  </script>
</head>

<body>
  <div align="center">
    <script type="text/javascript">
      roll1 = RandomInt(1, 6);
      roll2 = RandomInt(1, 6);

      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/comp519/Images/die" +
                     roll1 + ".gif'/>");
      document.write("  ");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/comp519/Images/die" +
                     roll2 + ".gif'/>");

    </script>
  </div>
</body>
</html>
```

view page

JavaScript Strings

- a class defines a new type (formally, Abstract Data Type)
  - encapsulates data (properties) and operations on that data (methods)
- a String encapsulates a sequence of characters, enclosed in quotes
  - properties include
    - length : stores the number of characters in the string
  - methods include
    - charAt(index) : returns the character stored at the given index (as in C++/Java, indices start at 0)
    - substring(start, end) : returns the part of the string between the start (inclusive) and end (exclusive) indices
    - toUpperCase() : returns copy of string with letters uppercase
    - toLowerCase() : returns copy of string with letters lowercase
  - to create a string, assign using new or just make a direct assignment (new is implicit)

    word = new String("foo"); word = "foo";
  - properties/methods are called exactly as in C++/Java

    word.length word.charAt(0)

String example: Palindromes

```
function Strip(str)
// Assumes: str is a string
// Returns: str with all but letters removed
{
  var copy = "";
  for (var i = 0; i < str.length; i++) {
    if ((str.charAt(i) >= "A" && str.charAt(i) <= "Z") ||
        (str.charAt(i) >= "a" && str.charAt(i) <= "z")) {
      copy += str.charAt(i);
    }
  }
  return copy;
}

function IsPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
  str = Strip(str.toUpperCase());

  for(var i = 0; i < Math.floor(str.length/2); i++) {
    if (str.charAt(i) != str.charAt(str.length-i-1)) {
      return false;
    }
  }
  return true;
}
```

suppose we want to test whether a word or phrase is a palindrome

noon Radar
Madam, I'm Adam.
A man, a plan, a canal: Panama!

must strip non-letters out of the word or phrase

make all chars uppercasein order to be case-insensitive

finally, traverse and compare chars from each end

```
<html>
<!-- COMP519  js09.html  16.08.2006 -->

<head>
 <title>Palindrome Checker</title>

  <script type="text/javascript">
    function Strip(str)
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }

    function IsPalindrome(str)
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }
  </script>
</head>

<body>
  <script type="text/javascript">
    text = prompt("Enter a word or phrase", "Madam, I'm Adam");

    if (IsPalindrome(text)) {
      document.write("'" + text + "' <b>is</b> a palindrome.");
    }
    else {
      document.write("'" + text + "' <b>is not</b> a palindrome.");
    }
  </script>
</body>
</html>
```

view page

JavaScript Arrays

- arrays store a sequence of items, accessible via an index
  - since JavaScript is loosely typed, elements do not have to be the same type
  - to create an array, allocate space using new (or can assign directly)

    items = new Array(10); // allocates space for 10 items
    items = new Array(); // if no size, will adjust dynamically
    items = [0,0,0,0,0,0,0,0,0,0]; // can assign size & values []
  - to access an array element, use [ ] (as in C++/Java)

    for (i = 0; i < 10; i++) {
      items[i] = 0; // stores 0 at each index
    }
  - the length property stores the number of items in the array

    for (i = 0; i < items.length; i++) {
      document.write(items[i] + "
"); // displays elements
    }

Array Example

```
<html>
<!-- COMP519  js10.html  16.08.2006 -->
<head>
 <title>Die Statistics</title>
 <script type="text/javascript"
src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/
random.js">
 </script>
</head>
<body>
 <script type="text/javascript">
    numRolls = 60000;
    dieSides = 6;

    rolls = new Array(dieSides+1);
    for (i = 1; i < rolls.length; i++) {
        rolls[i] = 0;
    }

    for(i = 1; i <= numRolls; i++) {
        rolls[RandomInt(1, dieSides)]++;
    }

    for (i = 1; i < rolls.length; i++) {
        document.write("Number of " + i + "'s = " +
                       rolls[i] + "<br />");
    }
  </script>
</body>
</html>
```

view page

suppose we want to simulate die rolls and verify even distribution

keep an array of counters:

initialize each count to 0

each time you roll X, increment `rolls[X]`

display each counter

Date Class

- String & Array are the most commonly used classes in JavaScript
  - other, special purpose classes & objects also exist
- the Date class can be used to access the date and time
  - to create a Date object, use new & supply year/month/day/… as desired

```
today = new Date();          // sets to current date & time
newYear = new Date(2002,0,1); //sets to Jan 1, 2002  12:00AM
```

  - methods include:

```
newYear.getYear()
newYear.getMonth()
newYear.getDay()
newYear.getHours()
newYear.getMinutes()
newYear.getSeconds()
newYear.getMilliseconds()
```

can access individual components of a date

Date Example

```
<html>
<!-- COMP519  js11.html  16.08.2006 -->

<head>
  <title>Time page</title>
</head>

<body>
  Time when page was loaded:
  <script type="text/javascript">
    now = new Date();

    document.write("<p>" + now + "</p>");

    time = "AM";
    hours = now.getHours();
    if (hours > 12) {
        hours -= 12;
        time = "PM"
    }
    else if (hours == 0) {
        hours = 12;
    }
    document.write("<p>" + hours + ":" +
                   now.getMinutes() + ":" +
                   now.getSeconds() + " " +
                   time + "</p>");
  </script>
</body>
</html>
```

by default, a date will be displayed in full, e.g.,

```
Sun Feb 03 22:55:20 GMT-0600
(Central Standard Time) 2002
```

can pull out portions of the date using the methods and display as desired

here, determine if "AM" or "PM" and adjust so hour between 1-12

```
10:55:20 PM
```

view page

Another Example

```
<html>
<!-- COMP519  js12.html  16.08.2006 -->

<head>
  <title>Time page</title>
</head>

<body>
  This year:
  <script type="text/javascript">
    now = new Date();
    newYear = new Date(2007,0,1);

    secs = Math.round((now-newYear)/1000);

    days = Math.floor(secs / 86400);
    secs -= days*86400;
    hours = Math.floor(secs / 3600);
    secs -= hours*3600;
    minutes = Math.floor(secs / 60);
    secs -= minutes*60

    document.write(days + " days, " +
                   hours + " hours, " +
                   minutes + " minutes, and " +
                   secs + " seconds.");
  </script>
</body>
</html>
```

you can add and subtract Dates:
the result is a number of milliseconds

here, determine the number of seconds since New Year's day

divide into number of days, hours, minutes and seconds

possible improvements?

view page

document Object

- Both IE and Netscape allow you to access information about an HTML document using the document object (Note: not a class!)

```
<html>
<!-- COMP519  js13.html  16.08.2006 -->

<head>
  <title>Documentation page</title>
</head>

<body>
  <table width="100%">
    <tr>
      <td><small><i>
        <script type="text/javascript">
            document.write(document.URL);
        </script>
      </i></small></td>
      <td style="text-align: right;"><small><i>
        <script type="text/javascript">
            document.write(document.lastModified);
        </script>
      </i></small></td>
    </tr>
  </table>
</body>
</html>
```

`document.write(…)`
method that displays text in the page

`document.URL`
property that gives the location of the HTML document

`document.lastModified`
property that gives the date & time the HTML document was saved

view page

navigator Object

`navigator.appName` property that gives the browser name

`navigator.appVersion` property that gives the browser version

```
<!-- MSIE.css  -->

a {text-decoration:none;
   font-size:larger;
   color:red;
   font-family:Arial}
a:hover {color:blue}
```

```
<!-- Netscape.css  -->

a {font-family:Arial;
   color:white;
   background-color:red}
```

```
<html>
<!-- COMP519  js14.html  16.08.2006 -->

<head>
  <title>Dynamic Style Page</title>
  <script type="text/javascript">
    if (navigator.appName == "Netscape") {
      document.write('<link rel=stylesheet '+
        'type="text/css"
href="Netscape.css">');
    }
    else {
      document.write('<link rel=stylesheet ' +
        'type="text/css" href="MSIE.css">');
    }
  </script>
</head>
<body>
Here is some text with a
<a href="javascript:alert('GO AWAY')">link</a>.
</body>
</html>
```

view page

User-Defined Classes

- can define new classes, but the notation is awkward
  - simply define a function that serves as a constructor
  - specify data fields & methods using this
  - no data hiding: can't protect data or methods

```
// COMP519       Die.js       16.08.2006//
// Die class definition
////////////////////////////////////////

function Die(sides)
{
   this.numSides = sides;
   this.numRolls = 0;
   this.Roll = Roll;
}

function Roll()
{
    this.numRolls++;
    return Math.floor(Math.random()*this.numSides) + 1;
}
```

define `Die` function (i.e., constructor)

initialize data fields in the function, preceded with this

similarly, assign method to separately defined function (which uses this to access data)

```
<html>
<!-- COMP519  js15.html  16.08.2006 -->

<head>
  <title>Dice page</title>

  <script type="text/javascript"
        src="Die.js">
  </script>
</head>

<body>
 <script type="text/javascript">
    die6 = new Die(6);
    die8 = new Die(8);

    roll6 = -1;    // dummy value to start loop
    roll8 = -2;    // dummy value to start loop
    while (roll6 != roll8) {
      roll6 = die6.Roll();
      roll8 = die8.Roll();

      document.write("6-sided: " + roll6 +
                     "    " +
                     "8-sided: " + roll8 + "<br />");
    }

    document.write("<br />Number of rolls: " +
                   die6.numRolls);
  </script>
</body>
</html>
```

view page

Class Example

create a Die object using new (similar to String and Array)

here, the argument to Die initializes numSides for that particular object

each Die object has its own properties (numSides & numRolls)

Roll(), when called on a particular Die, accesses its numSides property and updates its NumRolls

Advanced Javascript Issues

The XMLHttpRequest Object

- The XMLHttpRequest object is the developer's dream, because you can:
  - Update a web page with new data without reloading the page
  - Request data from a server after the page has loaded
  - Receive data from a server after the page has loaded
  - Send data to a server in the background
- The XMLHttpRequest object is supported in all modern browsers.

Creating an XMLHttpRequest Object

- Creating an XMLHttpRequest object is done with one single line of JavaScript.
  - var xmlhttp = new XMLHttpRequest()

Example

```
<script type="text/javascript">
var xmlhttp;
function loadXMLDoc(url)
{
xmlhttp=null;
if (window.XMLHttpRequest)
  {// code for all new browsers
  xmlhttp=new XMLHttpRequest();
  }
else if (window.ActiveXObject)
  {// code for IE5 and IE6
  xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
…
```


Example (Continued)

```
…
if (xmlhttp!=null)
  {
  xmlhttp.onreadystatechange=state_Change;
  xmlhttp.open("GET",url,true);
  xmlhttp.send(null);
  }
else
  {
  alert("Your browser does not support XMLHTTP.");
  }
}
```


Example (Continued)

```
…
function state_Change()
{
if (xmlhttp.readyState == 4 )
  {// 4 = "loaded"
  if (xmlhttp.status == 200)
    {// 200 = OK
    // ...our code here...
    }
  else
    {
    alert("Problem retrieving XML data");
    }
  }
}
</script>
```


Why Use Async=true?

- The previous example use "true" in the third parameter of open()
  - The parameter specifies whether the request should be handled asynchronously;
  - True means that the script continues to run after the send() method, without waiting for a response from the server.
  - The *onreadystatechange* event complicates the code. But it is the safest way if you want to prevent the code from stopping if you don't get a response from the server.
- By setting the parameter to "false", you can avoid the extra *onreadystatechange* code
  - Use this if it's not important to execute the rest of the code if the request fails.

Store a Cookie

```
function setCookie(c_name,value,expiredays)
{
  var exdate=new Date();
  exdate.setDate(exdate.getDate()+expiredays);
  document.cookie=c_name+ "=" +escape(value)+
  ((expiredays==null) ? "" :
  ";expires="+exdate.toGMTString());
  }
```


Check if a cookie has been set

```
function getCookie(c_name)
{
if (document.cookie.length>0)
  {
  c_start=document.cookie.indexOf(c_name + "=");
  if (c_start!=-1)
    {
    c_start=c_start + c_name.length+1;
    c_end=document.cookie.indexOf(";",c_start);
    if (c_end==-1) c_end=document.cookie.length;
    return unescape(document.cookie.substring(c_start,c_end));
    }
  }
return "";
```


Check for a Cookie

```
function checkCookie()
{
      username=getCookie('username');
      if (username!=null && username!="")
            {
            alert('Welcome again '+username+'!');
            }
      else
      {
      username=prompt('Please enter your name:',"");
      if (username!=null && username!="")
            {
            setCookie('username',username,365);
            }
      }
}
```


JavaScript Timing Events

- setTimeout()
  - Executes a code some time in the future
  - Ex. var t=setTimeout("javascript statement",milliseconds);
- clearTimeout()
  - cancels the setTimeout()

JavaScript Timing Events

```
<html>
<head>
<script type="text/javascript">
function timedMsg()
{
var t=setTimeout("alert('5 seconds!')",5000);
}
</script>
</head>

<body>
<form>
<input type="button" value="Display timed alertbox!"
onClick="timedMsg()">
</form>
</body>
```


Events

- onload and onUnload
  - Triggered when a use enters and leaves a page
- onFocus, onBlur and onChange
  - Used in combination with validation of form fields
  - Example:
    <input type="text" size="30" id="email" onchange="checkEmail()">
- onSubmit
  - used to validate ALL form fields before submitting it
  - Example:
    - <form method="post" action="xxx.htm" onsubmit="return checkForm()">
- onMouseOver and onMouseOut
  - Used to create "animated" buttons
  - Example

    <img src="w3schools.gif" width="100" height="30">

JavaScript Form Validation

```
function validate_required(field,alerttxt)
{
with (field)
{
  if (value==null||value=="")
  {
  alert(alerttxt);return false;
  }
  else
  {
  return true;
  }
}
}
```


JavaScript Form Validation

```
<html>
<head>
<script type="text/javascript">
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
  {alert(alerttxt);return false;}
else {return true}
}
}

function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,"Email must be filled out!")==false)
  {email.focus();return false;}
}
}
</script>
</head>

<body>
<form action="submitpage.htm"
```