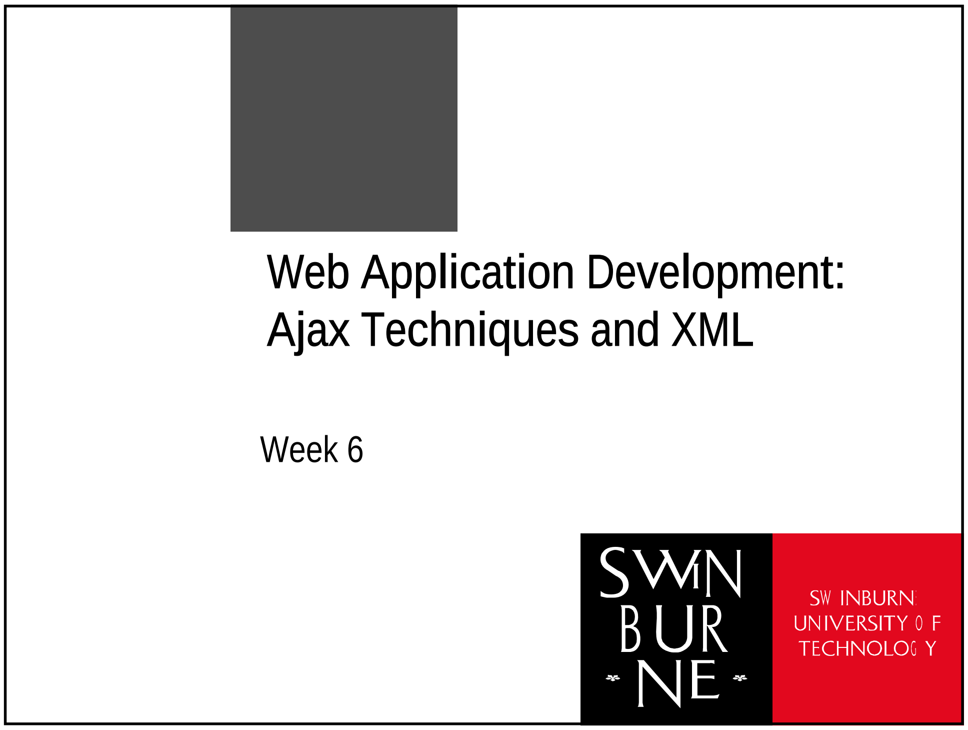
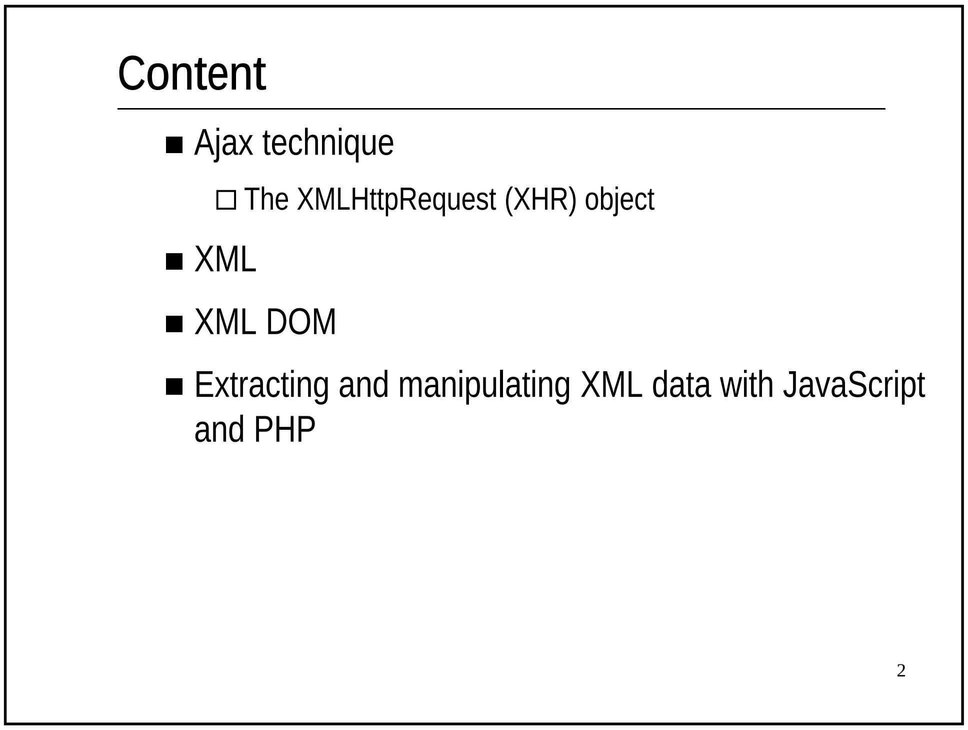


Web Application Development: Ajax Techniques and XML

Week 6

SWINBURNE UNIVERSITY OF TECHNOLOGY



Content

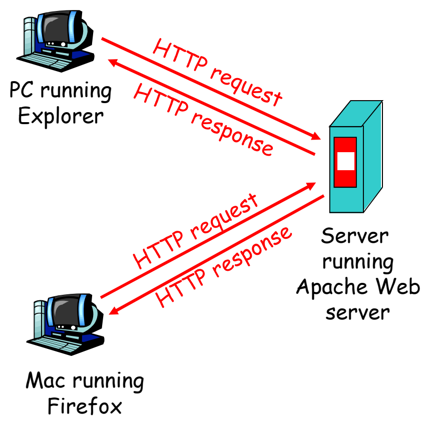
- Ajax technique
  - The XMLHttpRequest (XHR) object
- XML
- XML DOM
- Extracting and manipulating XML data with JavaScript and PHP

2

HTTP Overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
  - ☐ *client:* browser that requests, receives, "displays" Web objects
  - ☐ *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



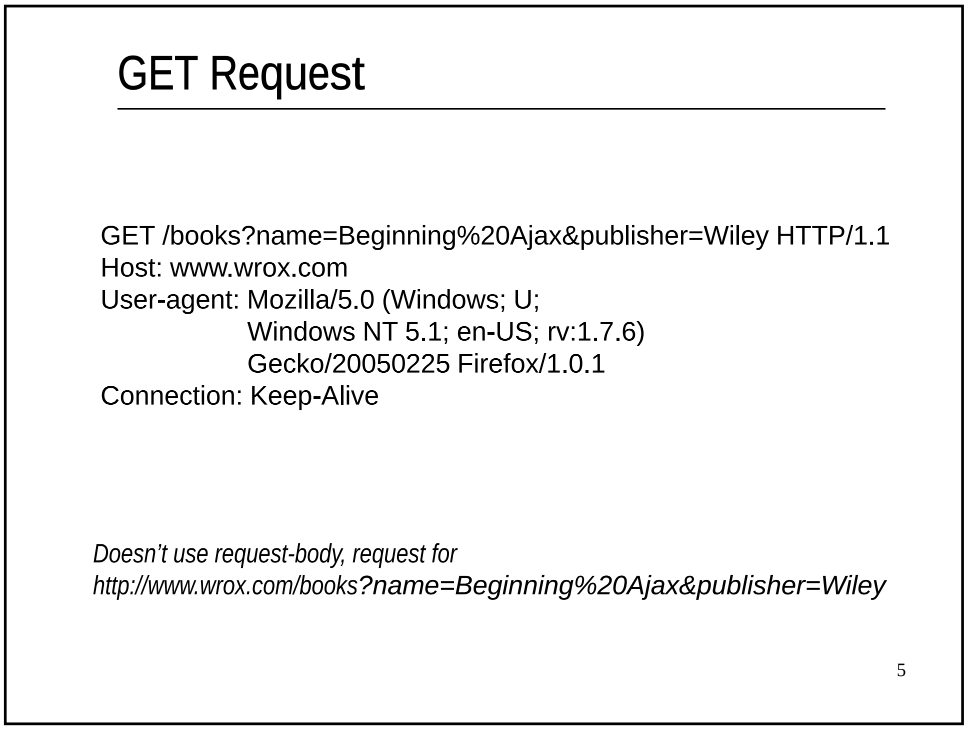
3

HTTP Requests

- HTTP request message format:

  <request-line>
  <headers>
  <blank-line>
  [<request-body>]
- two request types of interest
  - ☐GET
  - ☐POST

4

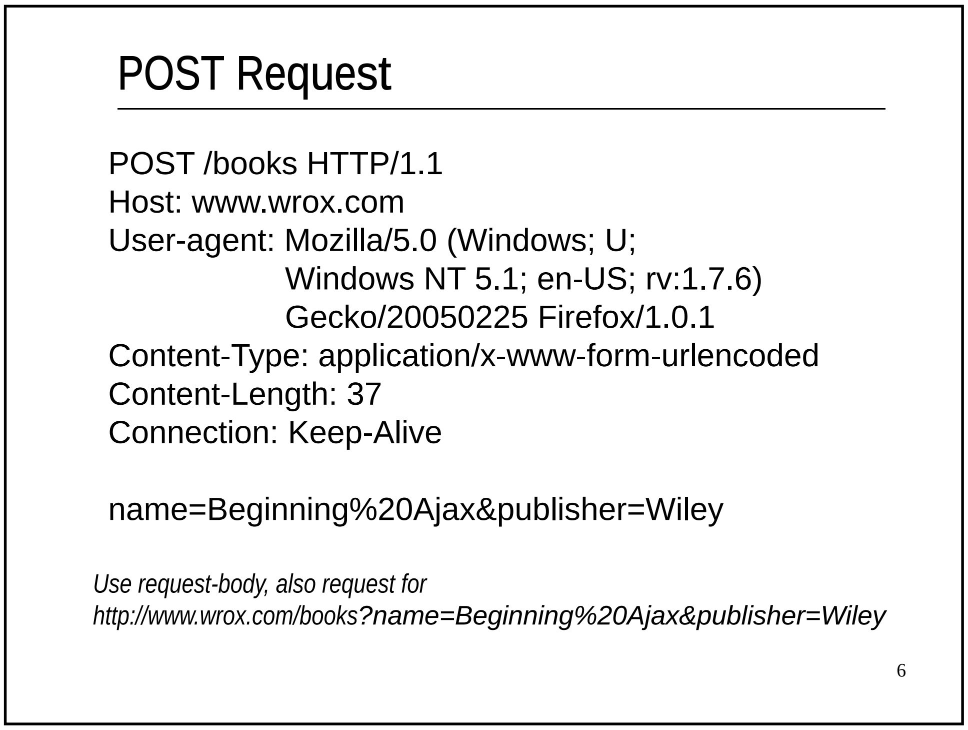


GET Request

```
GET /books?name=Beginning%20Ajax&publisher=Wiley HTTP/1.1
Host: www.wrox.com
User-agent: Mozilla/5.0 (Windows; U;
            Windows NT 5.1; en-US; rv:1.7.6)
            Gecko/20050225 Firefox/1.0.1
Connection: Keep-Alive
```

Doesn't use request-body, request for
http://www.wrox.com/books?name=Beginning%20Ajax&publisher=Wiley

5

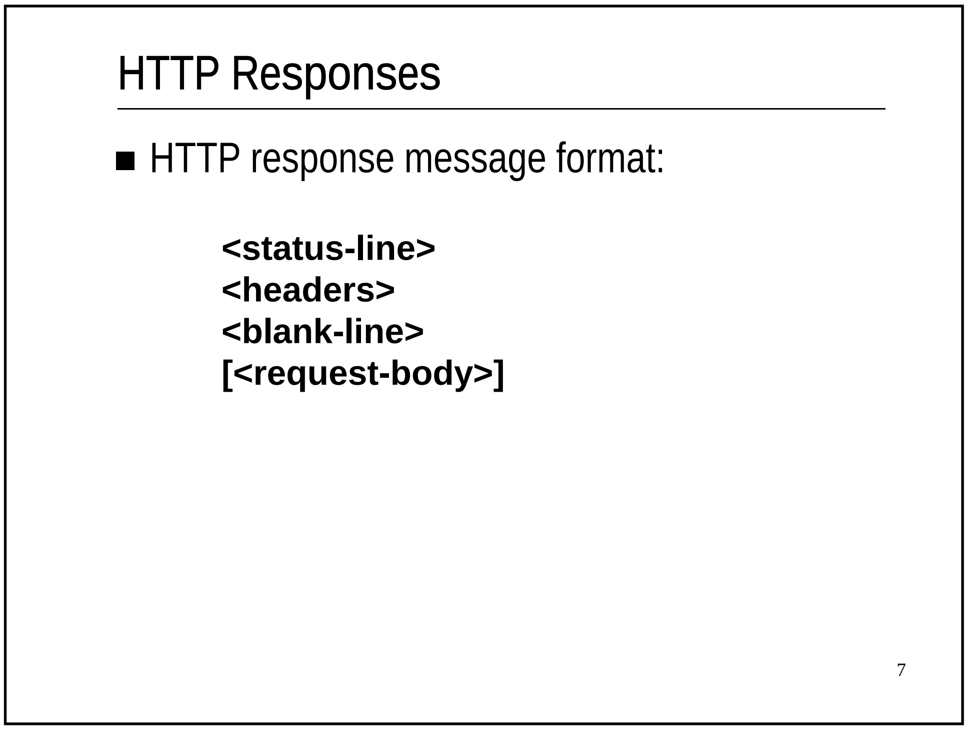


POST Request

```
POST /books HTTP/1.1
Host: www.wrox.com
User-agent: Mozilla/5.0 (Windows; U;
            Windows NT 5.1; en-US; rv:1.7.6)
            Gecko/20050225 Firefox/1.0.1
Content-Type: application/x-www-form-urlencoded
Content-Length: 37
Connection: Keep-Alive

name=Beginning%20Ajax&publisher=Wiley
```

Use request-body, also request for
http://www.wrox.com/books?name=Beginning%20Ajax&publisher=Wiley

6

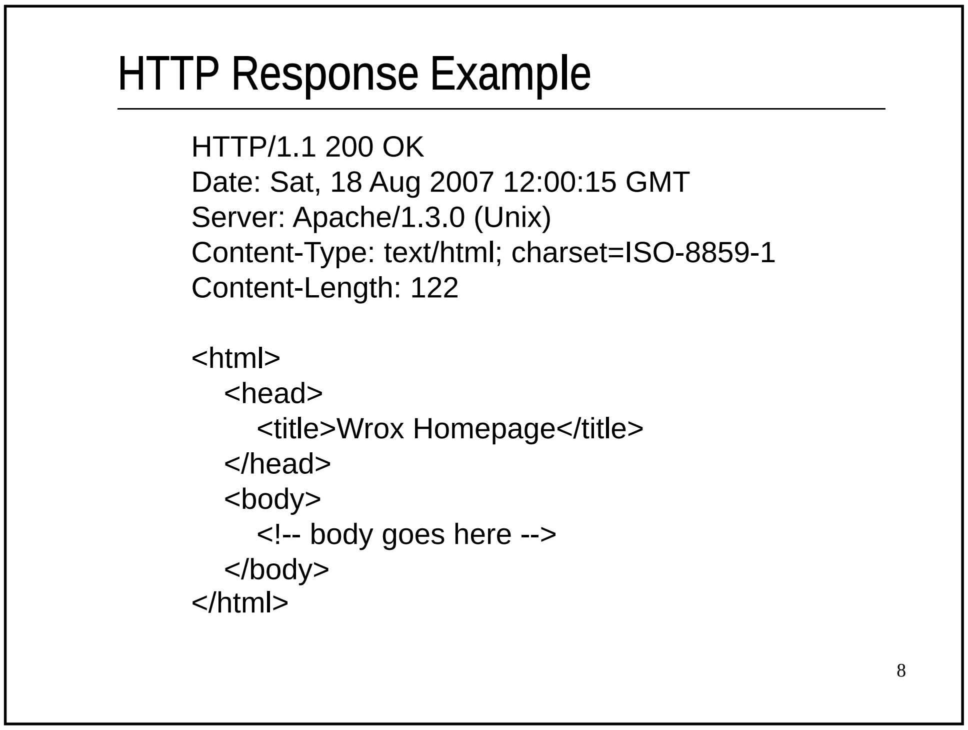


HTTP Responses

- HTTP response message format:

  <status-line>
  <headers>
  <blank-line>
  [<request-body>]

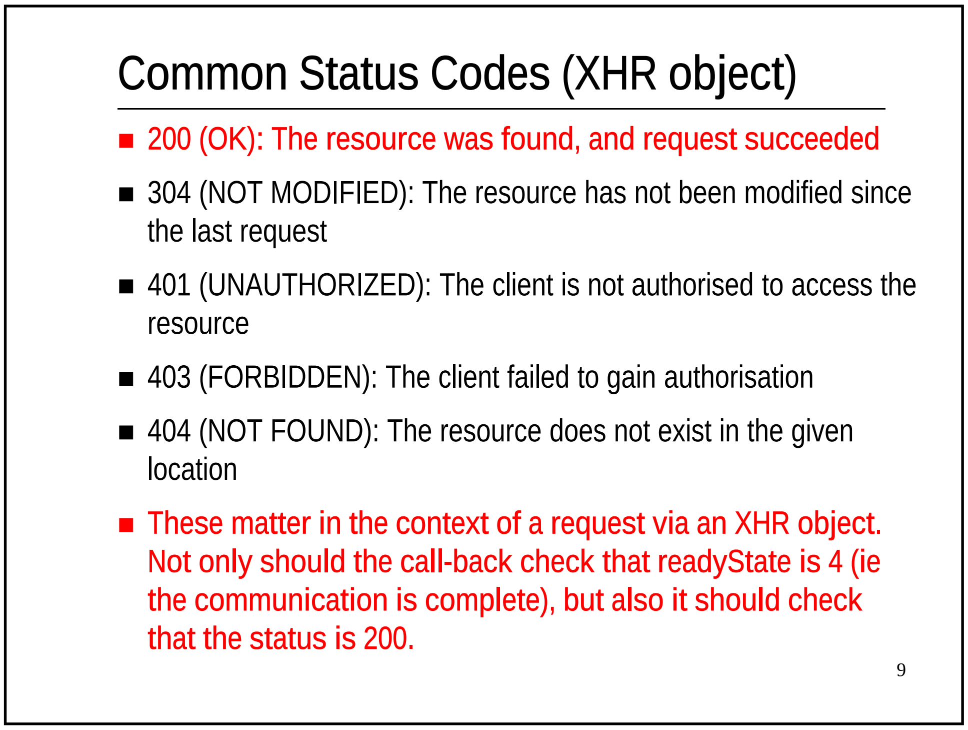
7



HTTP Response Example

```
HTTP/1.1 200 OK
Date: Sat, 18 Aug 2007 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Content-Type: text/html; charset=ISO-8859-1
Content-Length: 122

<html>
    <head>
        <title>Wrox Homepage</title>
    </head>
    <body>
        <!-- body goes here -->
    </body>
</html>
```

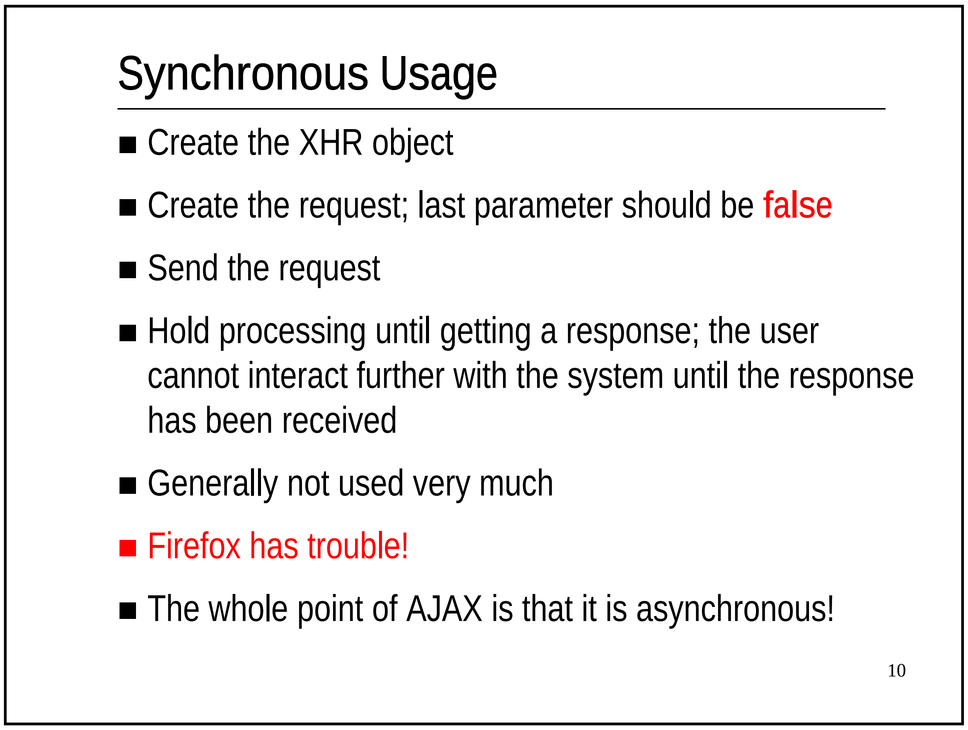
8



Common Status Codes (XHR object)

- 200 (OK): The resource was found, and request succeeded
- 304 (NOT MODIFIED): The resource has not been modified since the last request
- 401 (UNAUTHORIZED): The client is not authorised to access the resource
- 403 (FORBIDDEN): The client failed to gain authorisation
- 404 (NOT FOUND): The resource does not exist in the given location
- These matter in the context of a request via an XHR object. Not only should the call-back check that readyState is 4 (ie the communication is complete), but also it should check that the status is 200.

9



Synchronous Usage

- Create the XHR object
- Create the request; last parameter should be **false**
- Send the request
- Hold processing until getting a response; the user cannot interact further with the system until the response has been received
- Generally not used very much
- Firefox has trouble!
- The whole point of AJAX is that it is asynchronous!

10

Asynchronous Usage

- Create the XHR object
- Create the request; last parameter should be **true**
- Set the call-back function for the *readystatechange* event on the XHR object; in this function, the client processing should occur only when the readyState property of the XHR object has the value 4, and the status property has the value 200
- Send the request
- Continue client processing, only interrupting it when getting a response
- When the readyState property changes, the readystatechange event fires, and the call-back function executes. The values of readyState and status are checked, and if they are 4 / 200, the main client-side processing will take place.

11

The readyState Property

- This is a property of an XHR object. The possible values are
  - 0 – uninitialized
  - 1 – open
  - 2 – sent
  - 3 – receiving
  - 4 – loaded
- When an XHR object is created, its readyState property has the value 0. As processing continues, the property will actually take on all the values 0,1,2,3,4 in succession. Each time it changes, the onreadystatechange event fires, so the call-back function will generally be called 4 times before the value reaches 4 (ie, the value “loaded”) .

12

Successful communication

- Communication between an XHR object xhr and the server concludes when
  - xhr.readyState is 4
  - xhr.status is 200

  then the main processing on the client take place
- As noted, xhr.readyState changes several times during the communication. There is only a point in lookimg at xhr.status once xhr.readyState reaches the value 4. It will then be normal for xhr.status to be 200 – only when something has gone wrong with the data transmission will this not be the case

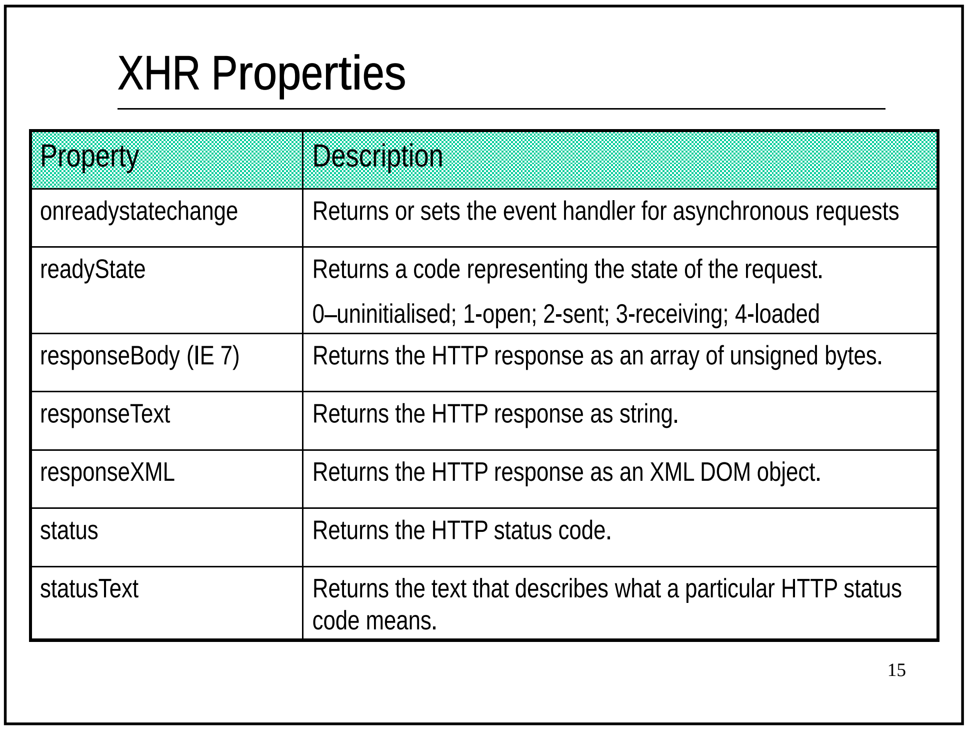
13

Successful communication

The two approaches to coding in the call-back function are as follows:

```
if ((xhr.readyState == 4) && (xhr.status == 200))
            // processing here
```

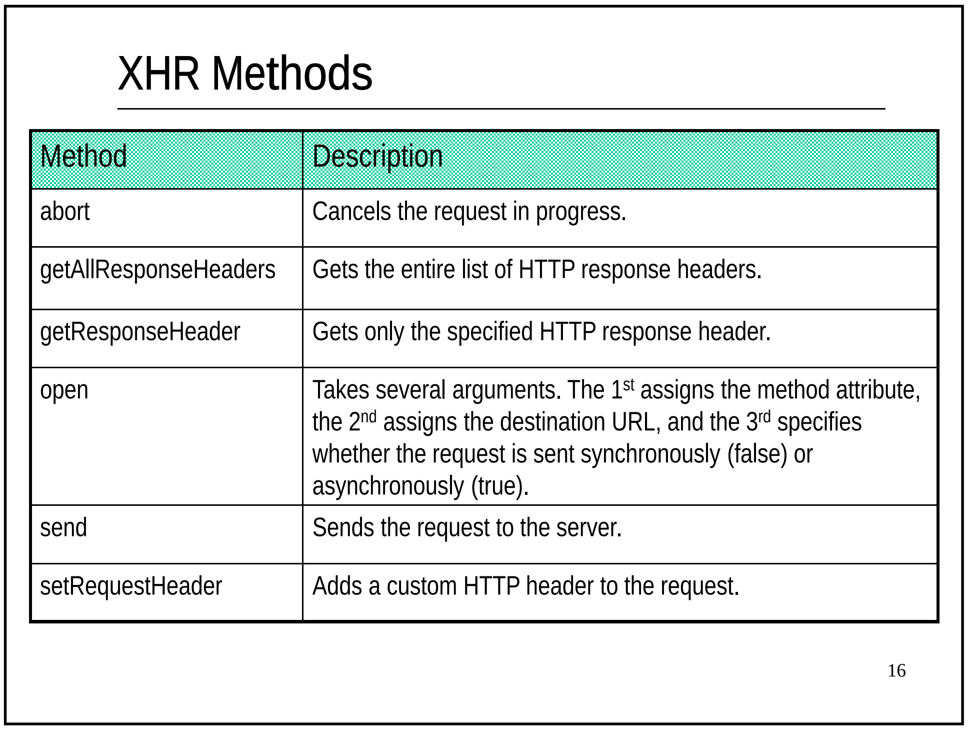
```
if (xhr.readyState == 4)
{
   if (xhr.status == 200)
   {
       // processing for successful communication here
    }
    else
    {
       // manage error case where status is not 200
    }
}
```



XHR Properties

Property	Description
onreadystatechange	Returns or sets the event handler for asynchronous requests
readyState	Returns a code representing the state of the request. 0–uninitialised; 1-open; 2-sent; 3-receiving; 4-loaded
responseBody (IE 7)	Returns the HTTP response as an array of unsigned bytes.
responseText	Returns the HTTP response as string.
responseXML	Returns the HTTP response as an XML DOM object.
status	Returns the HTTP status code.
statusText	Returns the text that describes what a particular HTTP status code means.

15



XHR Methods

Method	Description
abort	Cancels the request in progress.
getAllResponseHeaders	Gets the entire list of HTTP response headers.
getResponseHeader	Gets only the specified HTTP response header.
open	Takes several arguments. The 1st assigns the method attribute, the 2nd assigns the destination URL, and the 3rd specifies whether the request is sent synchronously (false) or asynchronously (true).
send	Sends the request to the server.
setRequestHeader	Adds a custom HTTP header to the request.

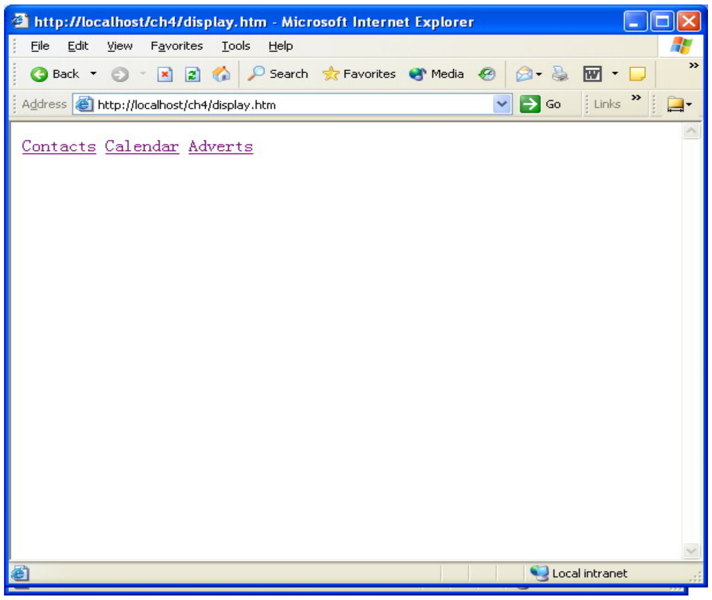
16

Example

- Here we develop a simple web page which allows the user to display a list of their contacts, their calendar, and a list of adverts that are maintained on the server
- There is no database maintenance – this system just gives access to data on the server which is to be displayed
- The user selects contacts, calendar or adverts by clicking on a link
- Three separate *divs* are set up in advance in the client-side page to receive the three types of response data. Whatever item is selected by the user, details will be displayed in the appropriate place on the page
- We do not bother here about styling, but could use a CSS to control where the different types of data might be positioned, and how they might appear

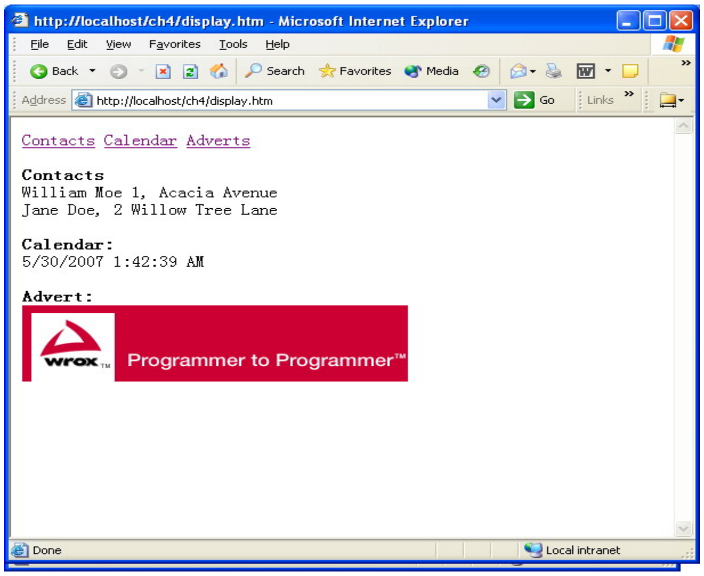
17

Using XMLHttpRequest for Dynamic Displaying


http://localhost/ch4/display.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch4/display.htm Go Links
Contacts Calendar Adverts
Local intranet

18

Clicking Adverts


http://localhost/ch4/display.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch4/display.htm
Go Links
Done
Local intranet
wrox
Programmer to Programmer™

Contacts Calendar Adverts

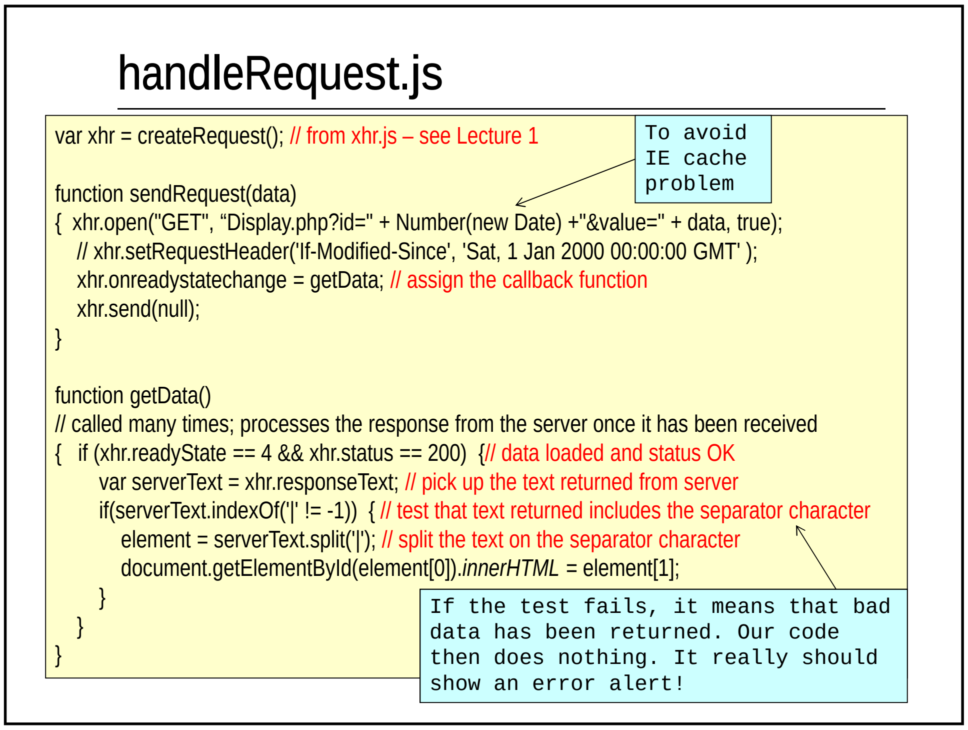
```
Contacts
William Moe 1, Acacia Avenue
Jane Doe, 2 Willow Tree Lane

Calendar:
5/30/2007 1:42:39 AM

Advert:
```

19

Display.htm

```
<html>
<head>
    <script type="text/javascript" src="xhr.js"></script>
    <script type="text/javascript" src="handleRequest.js"></script>
</head>
<body>
   <a href="#" onclick ="sendRequest('Contacts');return false;">Contacts</a>
   <a href="#" onclick ="sendRequest('Calendar');return false;">Calendar</a>
   <a href="#" onclick ="sendRequest('Adverts');return false;">Adverts</a>
   <br/>
   <div id="box1">
   </div>
   <div id="box2">
   </div>
   <div id="box3">
   </div>
</body>
</html>
```

20

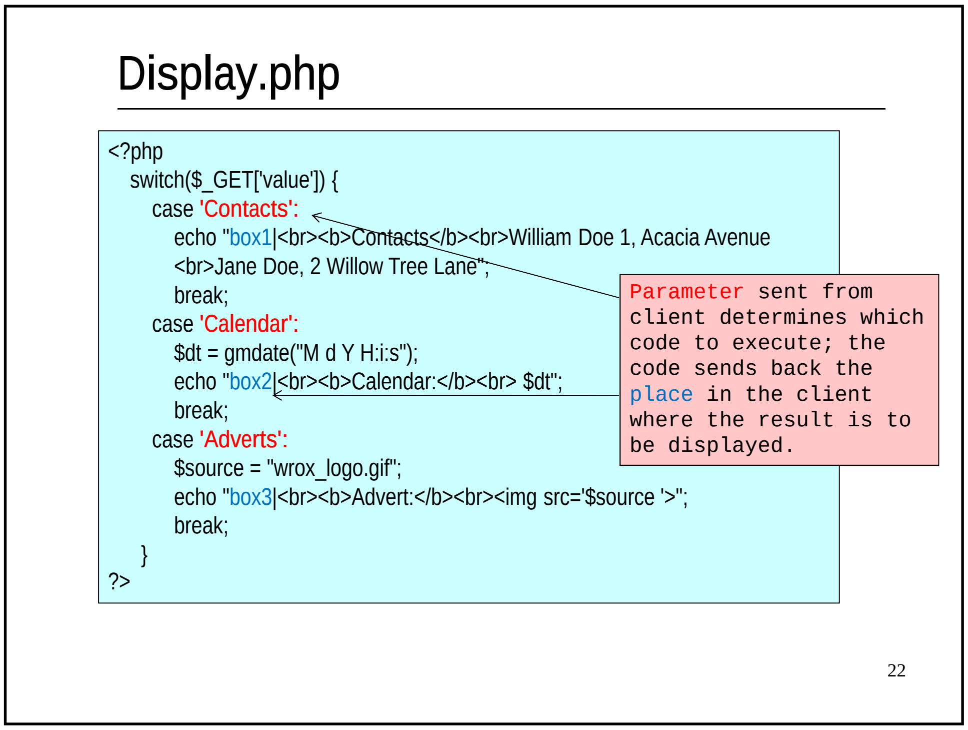


handleRequest.js

```
var xhr = createRequest(); // from xhr.js – see Lecture 1

function sendRequest(data)
{  xhr.open("GET", "Display.php?id=" + Number(new Date) +"&value=" + data, true);
   // xhr.setRequestHeader('If-Modified-Since', 'Sat, 1 Jan 2000 00:00:00 GMT' );
   xhr.onreadystatechange = getData; // assign the callback function
   xhr.send(null);
}

function getData()
// called many times; processes the response from the server once it has been received
{   if (xhr.readyState == 4 && xhr.status == 200)  {// data loaded and status OK
       var serverText = xhr.responseText; // pick up the text returned from server
       if(serverText.indexOf('|' != -1))  { // test that text returned includes the separator character
          element = serverText.split('|'); // split the text on the separator character
          document.getElementById(element[0]).innerHTML = element[1];
       }
    }
}
```

To avoid IE cache problem

If the test fails, it means that bad data has been returned. Our code then does nothing. It really should show an error alert!

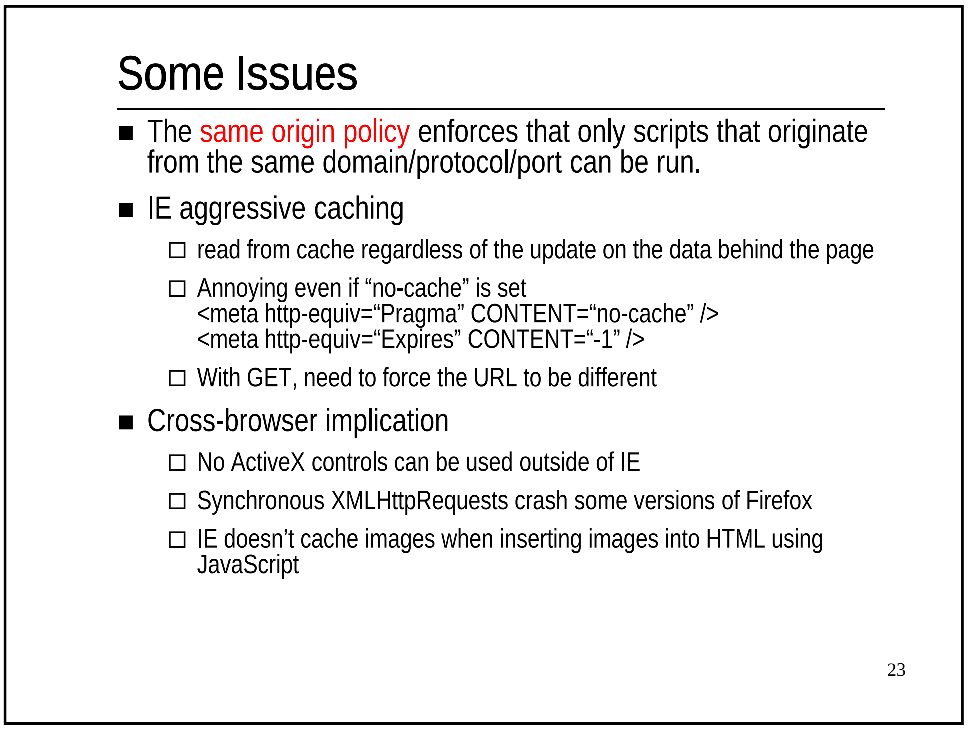


Display.php

```
<?php
   switch($_GET['value']) {
      case 'Contacts':
         echo "box1|<br><b>Contacts</b><br>William Doe 1, Acacia Avenue
         <br>Jane Doe, 2 Willow Tree Lane";
         break;
      case 'Calendar':
         $dt = gmdate("M d Y H:i:s");
         echo "box2|<br><b>Calendar:</b><br> $dt";
         break;
      case 'Adverts':
         $source = "wrox_logo.gif";
         echo "box3|<br><b>Advert:</b><br><img src='$source '>";
         break;
   }
?>
```

Parameter sent from client determines which code to execute; the code sends back the place in the client where the result is to be displayed.

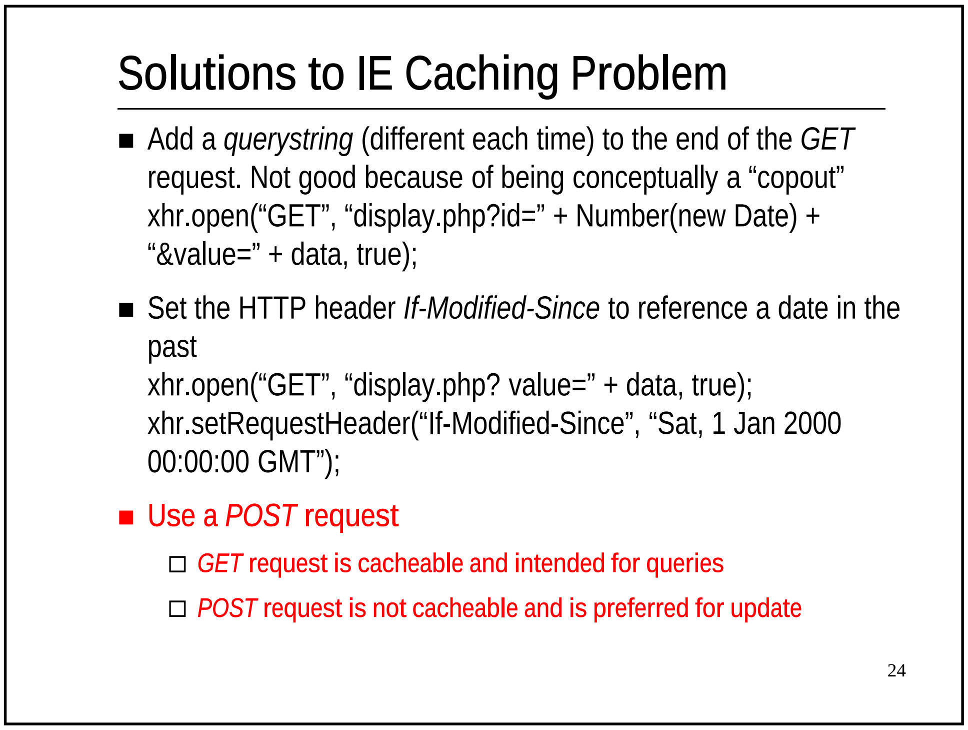
22



Some Issues

- The same origin policy enforces that only scripts that originate from the same domain/protocol/port can be run.
- IE aggressive caching
  - read from cache regardless of the update on the data behind the page
  - Annoying even if “no-cache” is set
    <meta http-equiv=“Pragma” CONTENT=“no-cache” />
    <meta http-equiv=“Expires” CONTENT=“-1” />
  - With GET, need to force the URL to be different
- Cross-browser implication
  - No ActiveX controls can be used outside of IE
  - Synchronous XMLHttpRequests crash some versions of Firefox
  - IE doesn’t cache images when inserting images into HTML using JavaScript

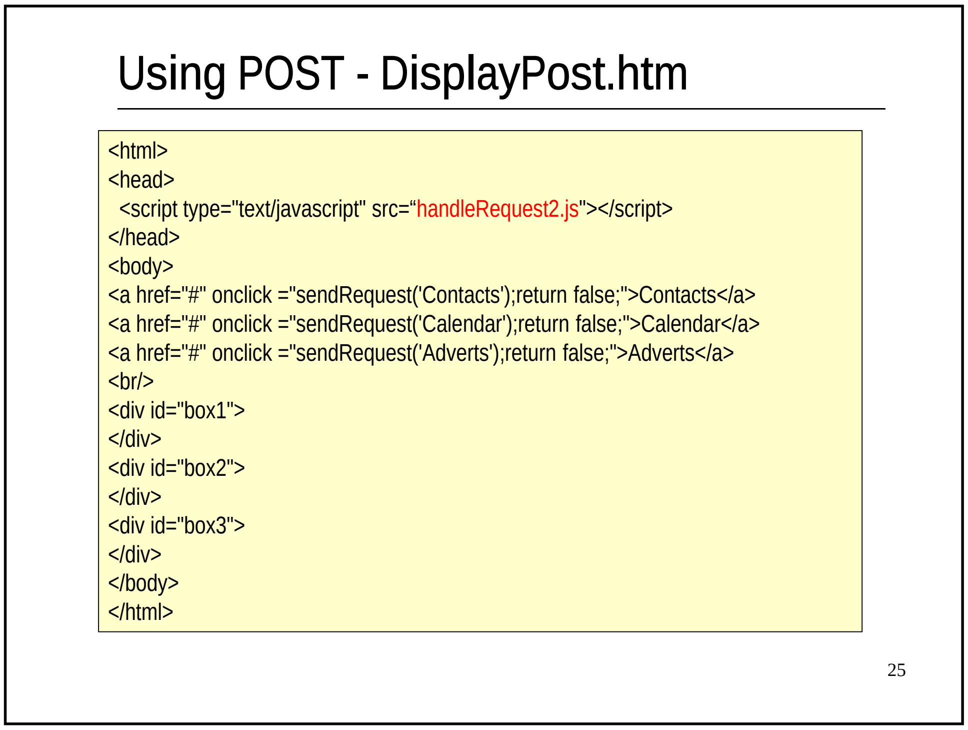
23



Solutions to IE Caching Problem

- Add a *querystring* (different each time) to the end of the *GET* request. Not good because of being conceptually a “copout”
  xhr.open(“GET”, “display.php?id=” + Number(new Date) + “&value=” + data, true);
- Set the HTTP header *If-Modified-Since* to reference a date in the past
  xhr.open(“GET”, “display.php? value=” + data, true);
  xhr.setRequestHeader(“If-Modified-Since”, “Sat, 1 Jan 2000 00:00:00 GMT”);
- Use a *POST* request
  - *GET* request is cacheable and intended for queries
  - *POST* request is not cacheable and is preferred for update

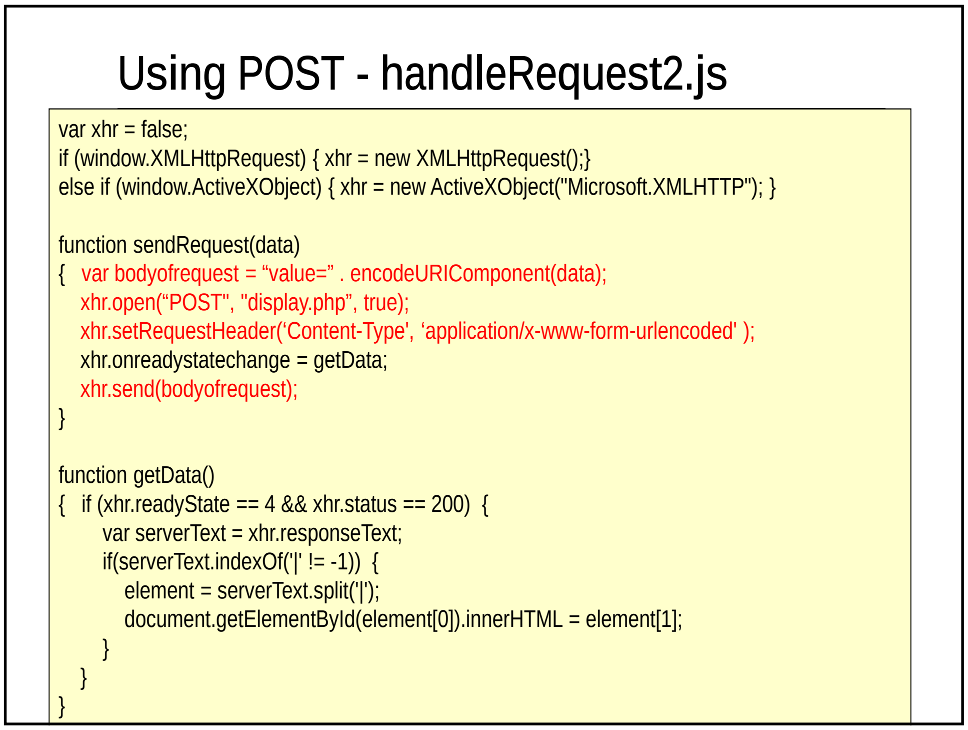
24



Using POST - DisplayPost.htm

```
<html>
<head>
  <script type="text/javascript" src="handleRequest2.js"></script>
</head>
<body>
<a href="#" onclick ="sendRequest('Contacts');return false;">Contacts</a>
<a href="#" onclick ="sendRequest('Calendar');return false;">Calendar</a>
<a href="#" onclick ="sendRequest('Adverts');return false;">Adverts</a>
<br/>
<div id="box1">
</div>
<div id="box2">
</div>
<div id="box3">
</div>
</body>
</html>
```

25



Using POST - handleRequest2.js

```
var xhr = false;
if (window.XMLHttpRequest) { xhr = new XMLHttpRequest();}
else if (window.ActiveXObject) { xhr = new ActiveXObject("Microsoft.XMLHTTP"); }

function sendRequest(data)
{   var bodyofrequest = "value=" . encodeURIComponent(data);
    xhr.open("POST", "display.php", true);
    xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded' );
    xhr.onreadystatechange = getData;
    xhr.send(bodyofrequest);
}

function getData()
{   if (xhr.readyState == 4 && xhr.status == 200)  {
        var serverText = xhr.responseText;
        if(serverText.indexOf('|' != -1))  {
            element = serverText.split('|');
            document.getElementById(element[0]).innerHTML = element[1];
        }
    }
}
```

XML

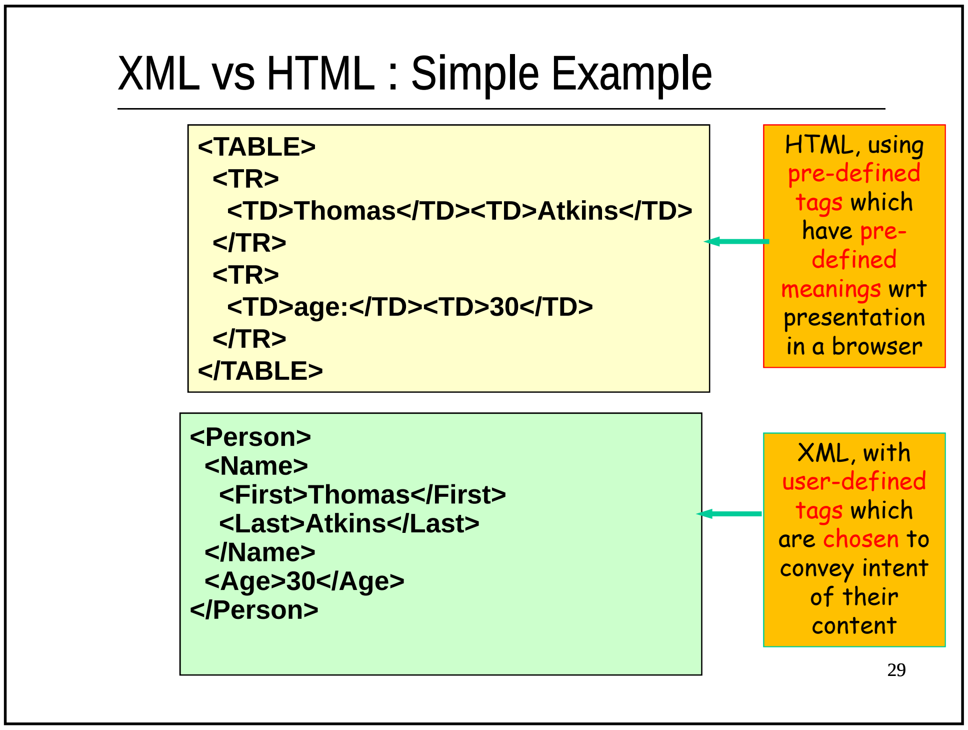
- XML is a standard for data representation and exchange
  – W3C XML 1.0 and 1.1 http://www.w3.org/XML/
- It provides a common format for expressing both data structures and contents
- Markup language for structured information
- A markup language is a set of symbols that can be placed in the text of a document to create boundaries and label the parts of that document
- XML is a meta-markup language because it lets you create your own markup language
- XML allows author to *customise* own elements, i.e., their own tags
- The elements of the author's own choice are used to structure data and provide it meaning

27

XML vs HTML

- HTML has fixed tag semantics (defined by browser behaviour) and pre-defined tags
  - <H1>, <P>, <BODY>, <TD> …
  - W3C keeps extending this tag set and semantics (ie, what happens when displayed in a browser)
- XML specifies neither a tag set nor semantics
  - It is a meta-language for describing (markup and other) languages
  - You can define your own tags, and the structure of these tags
  - Semantics are provided by applications that process XML documents or by style-sheets

28



XML vs HTML : Simple Example

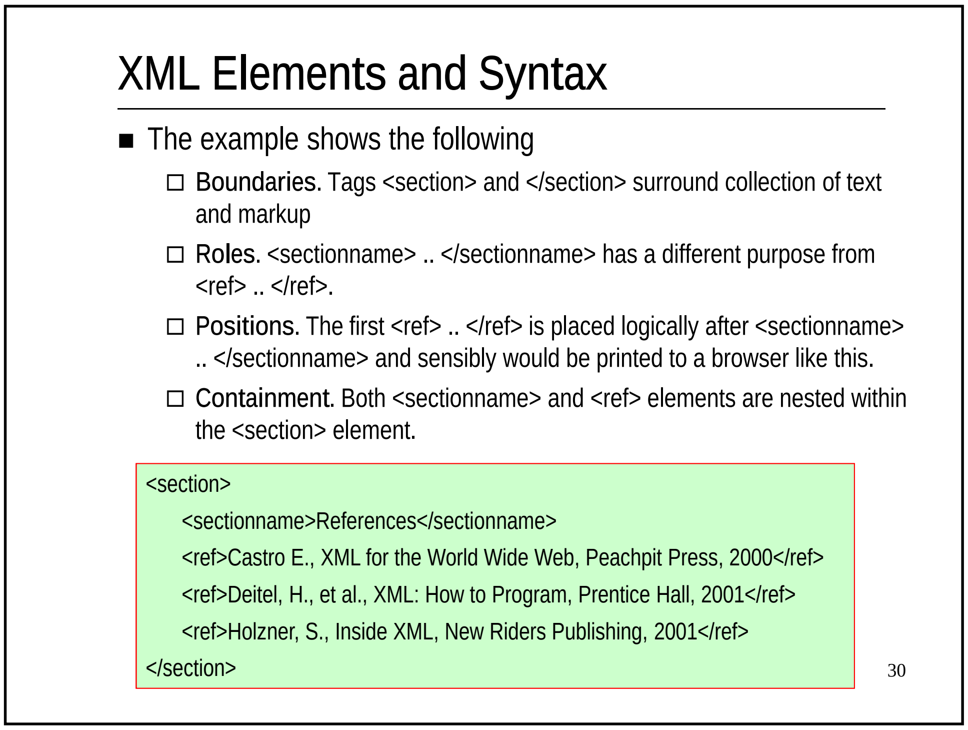
```
<TABLE>
  <TR>
    <TD>Thomas</TD><TD>Atkins</TD>
  </TR>
  <TR>
    <TD>age:</TD><TD>30</TD>
  </TR>
</TABLE>
```

HTML, using pre-defined tags which have pre-defined meanings wrt presentation in a browser

```
<Person>
  <Name>
    <First>Thomas</First>
    <Last>Atkins</Last>
  </Name>
  <Age>30</Age>
</Person>
```

XML, with user-defined tags which are chosen to convey intent of their content

29

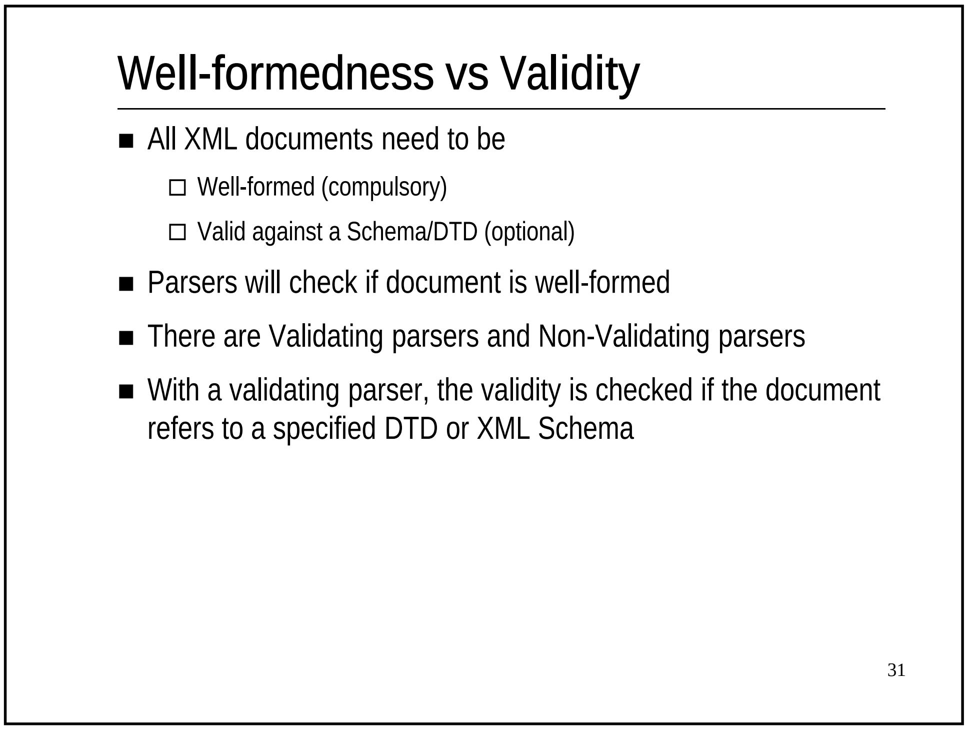


XML Elements and Syntax

- The example shows the following
  - Boundaries. Tags <section> and </section> surround collection of text and markup
  - Roles. <sectionname> .. </sectionname> has a different purpose from <ref> .. </ref>.
  - Positions. The first <ref> .. </ref> is placed logically after <sectionname> .. </sectionname> and sensibly would be printed to a browser like this.
  - Containment. Both <sectionname> and <ref> elements are nested within the <section> element.

```
<section>
    <sectionname>References</sectionname>
    <ref>Castro E., XML for the World Wide Web, Peachpit Press, 2000</ref>
    <ref>Deitel, H., et al., XML: How to Program, Prentice Hall, 2001</ref>
    <ref>Holzner, S., Inside XML, New Riders Publishing, 2001</ref>
</section>
```

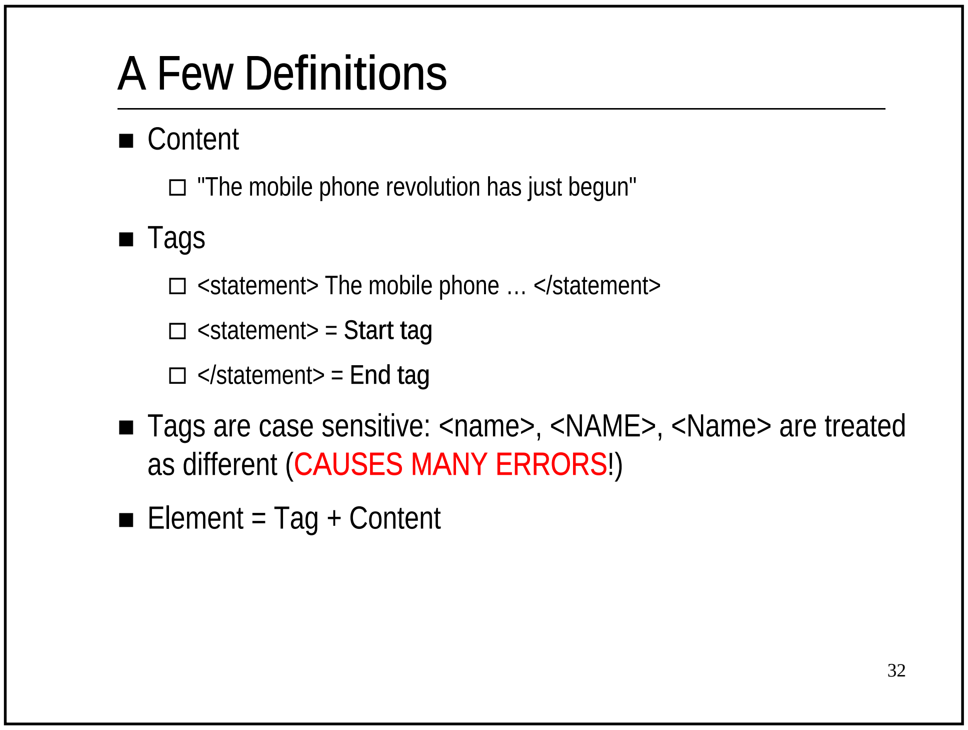
30



Well-formedness vs Validity

- All XML documents need to be
  - Well-formed (compulsory)
  - Valid against a Schema/DTD (optional)
- Parsers will check if document is well-formed
- There are Validating parsers and Non-Validating parsers
- With a validating parser, the validity is checked if the document refers to a specified DTD or XML Schema

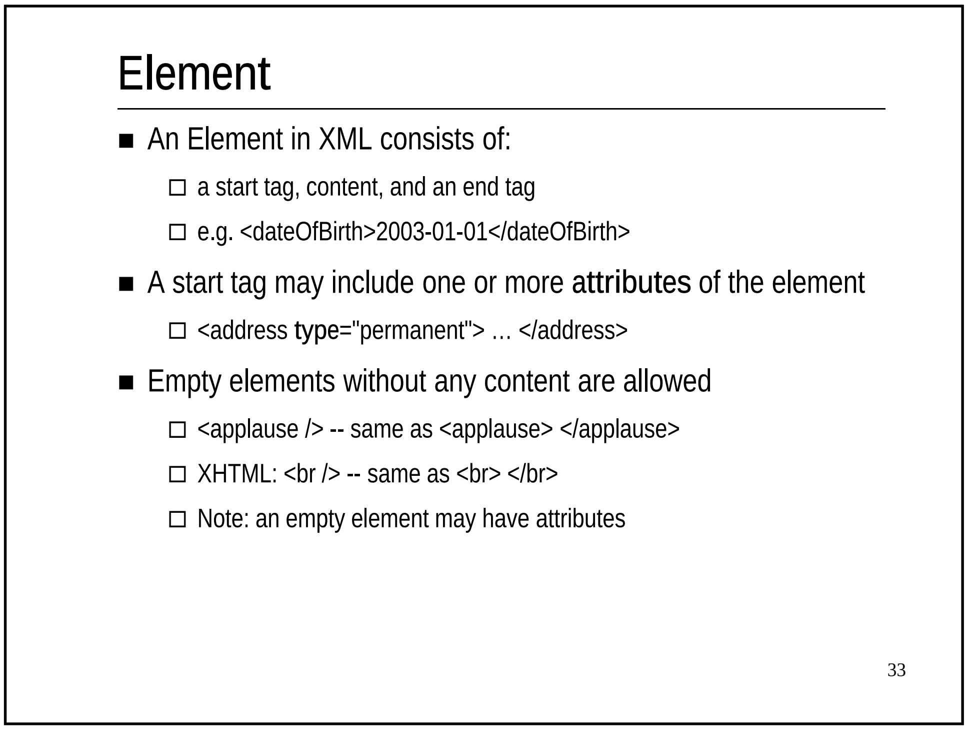
31



A Few Definitions

- Content
  - "The mobile phone revolution has just begun"
- Tags
  - <statement> The mobile phone … </statement>
  - <statement> = **Start tag**
  - </statement> = **End tag**
- Tags are case sensitive: <name>, <NAME>, <Name> are treated as different (CAUSES MANY ERRORS!)
- Element = Tag + Content

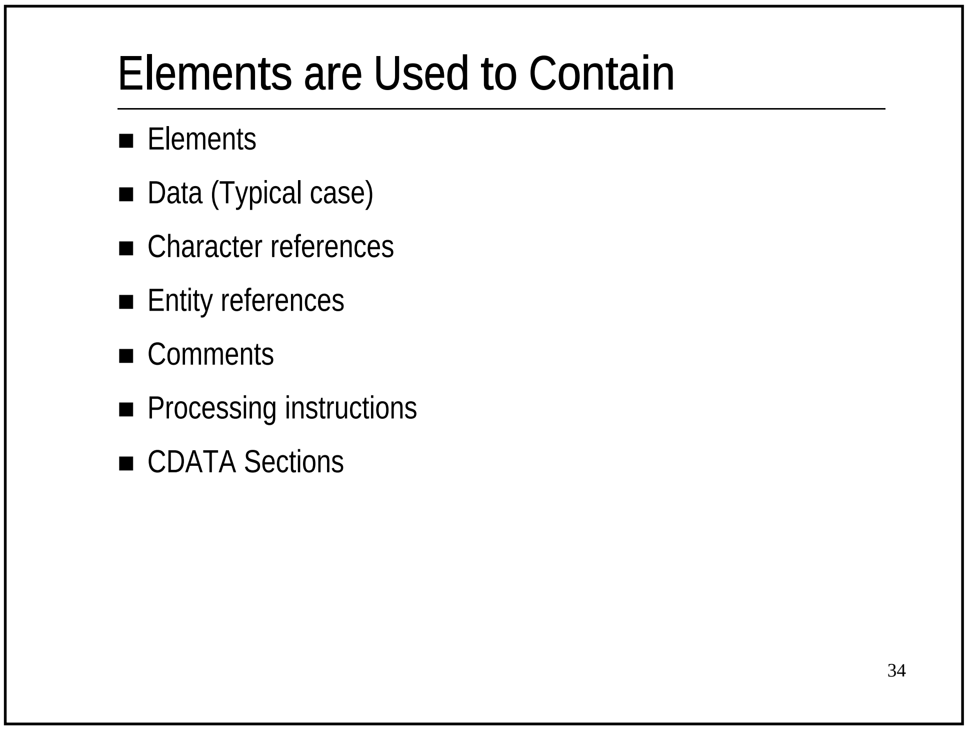
32



Element

- An Element in XML consists of:
  - a start tag, content, and an end tag
  - e.g. <dateOfBirth>2003-01-01</dateOfBirth>
- A start tag may include one or more **attributes** of the element
  - <address **type**="permanent"> … </address>
- Empty elements without any content are allowed
  - <applause /> **--** same as <applause> </applause>
  - XHTML: <br /> **--** same as <br> </br>
  - Note: an empty element may have attributes

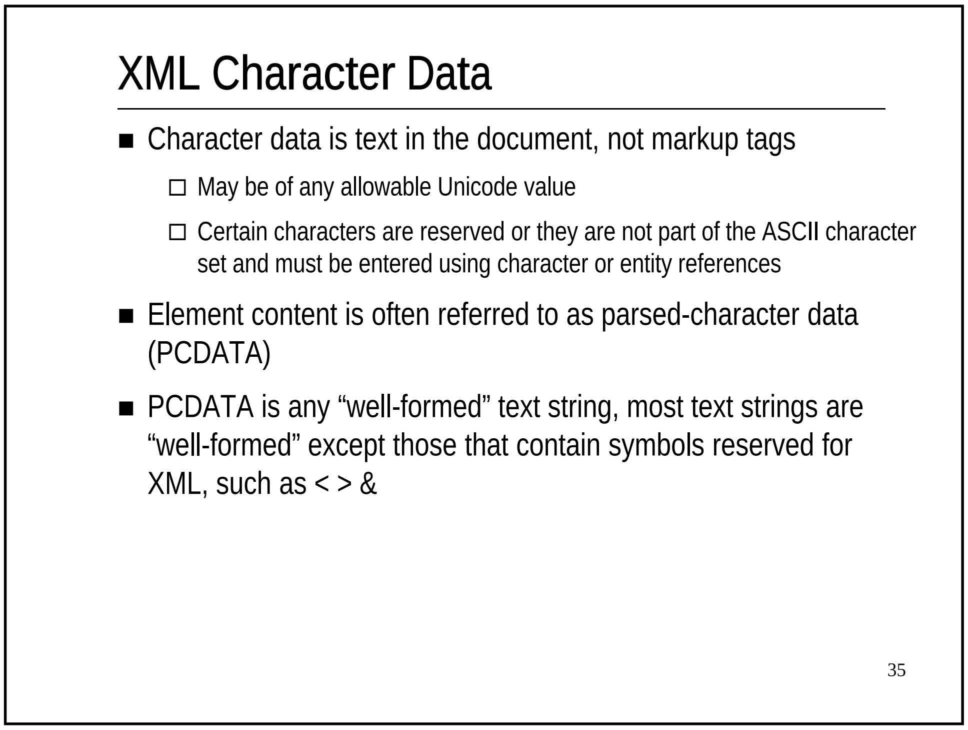
33



Elements are Used to Contain

- Elements
- Data (Typical case)
- Character references
- Entity references
- Comments
- Processing instructions
- CDATA Sections

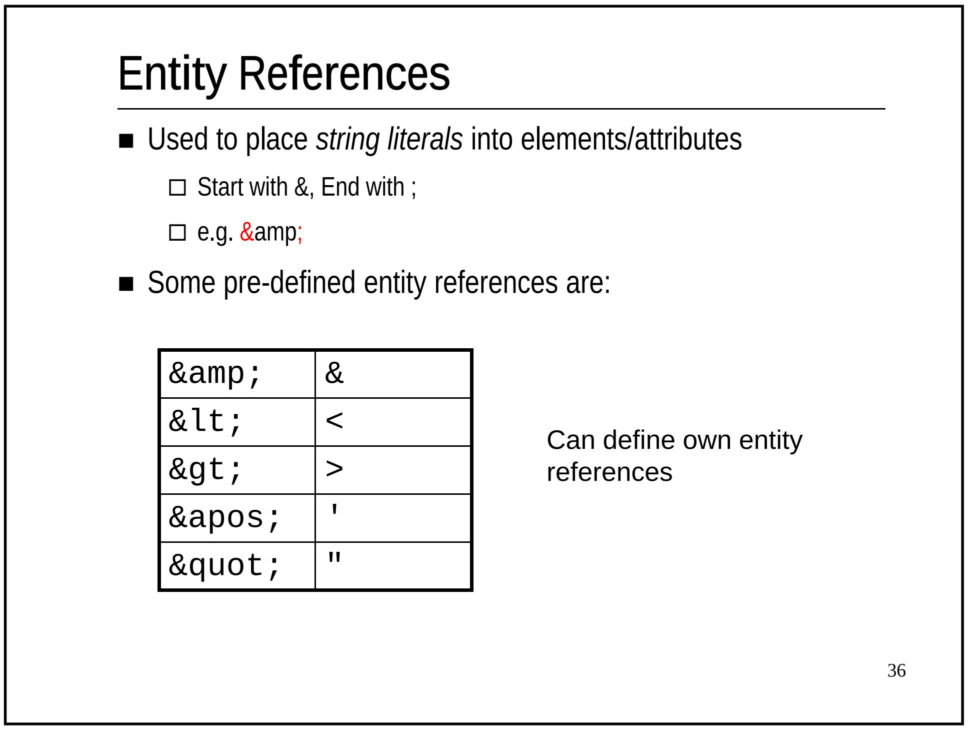
34



XML Character Data

- Character data is text in the document, not markup tags
  - May be of any allowable Unicode value
  - Certain characters are reserved or they are not part of the ASCII character set and must be entered using character or entity references
- Element content is often referred to as parsed-character data (PCDATA)
- PCDATA is any “well-formed” text string, most text strings are “well-formed” except those that contain symbols reserved for XML, such as < > &

35



Entity References

- Used to place *string literals* into elements/attributes
  - Start with &, End with ;
  - e.g. &amp;
- Some pre-defined entity references are:

`&amp;`	&
`&lt;`	<
`&gt;`	>
`&apos;`	'
`&quot;`	"

Can define own entity references

36

Character References

- A *character reference* is where use is made of a decimal or hexadecimal number to represent a character able to be stored in XML data and be displayable (for instance, in a Web browser), e.g., ©
- Such a character is not able to be placed in its displayable form into a document because it is not available from the input device, e.g., a Japanese character trying to be entered in a Turkish word processor
- Allows to represent Unicode characters
- Allowed forms of references are:
  - Decimal: &#DDDDD; (1 to 5 digits), e.g., &#169;
  - Hexadecimal: &#xHHHH; (1 to 4 digits), e.g., &#xA9;
  - Both the above represent ©
- Hexadecimal form is preferred
- Cannot use character references for element / attribute names

37

Comments, PIs and CDATA

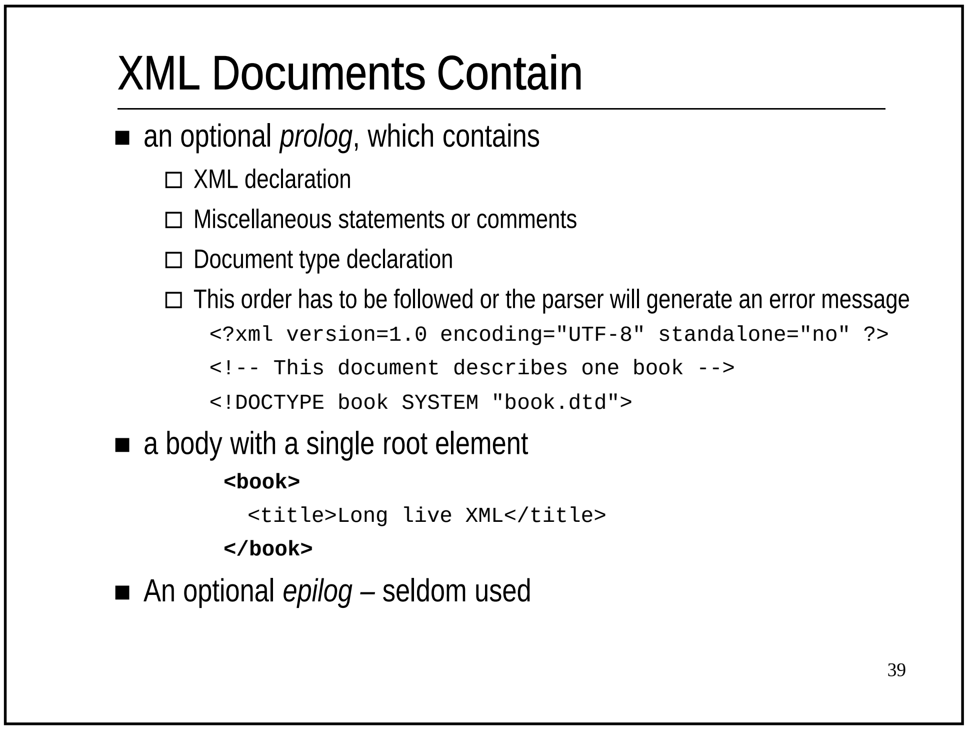
- Comments are like in HTML
  - **`<!-- This is a comment -->`**
- Processing Instructions (PIs) - processing hint, script code or presentational info can be indicated to a parser
  - <? Some processing instruction ?>
  - is parser-specific, so not all parsers will respond to a PI's in the same way
  - Examples:

```
<?xml-stylesheet href=“style.css” type=“text/css”?>
```

- CDATA sections are used to capture any data that may confuse the parser. They will not be parsed
  - Example:

```
<![CDATA[
  <html><body>Content</body></html>
]]>
```

38



XML Documents Contain

- an optional *prolog*, which contains
  - XML declaration
  - Miscellaneous statements or comments
  - Document type declaration
  - This order has to be followed or the parser will generate an error message

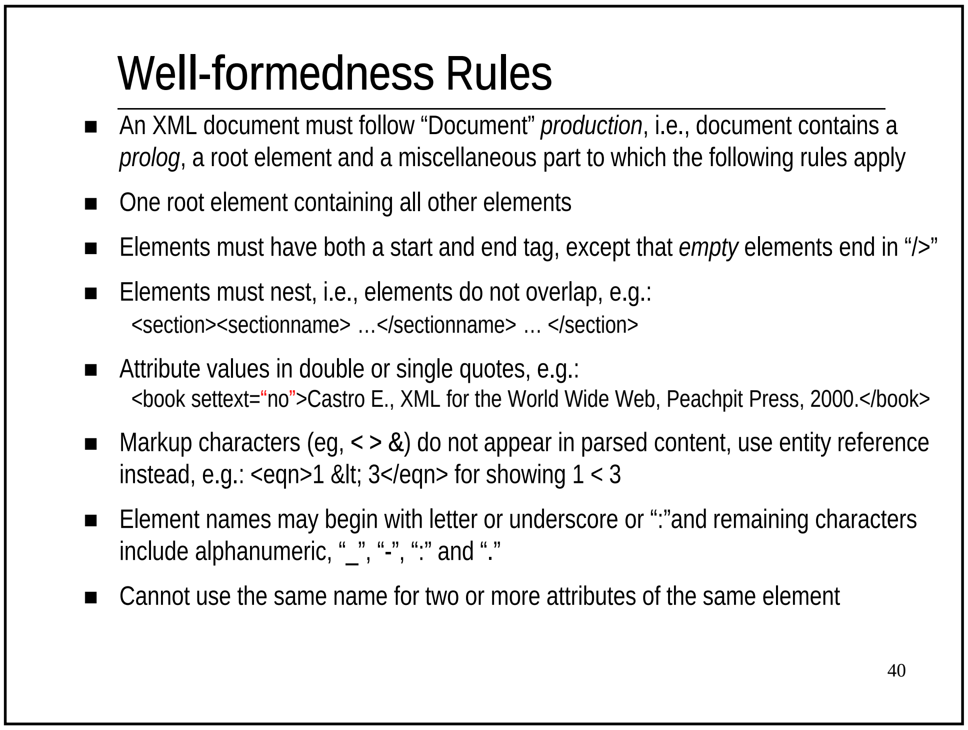
```
<?xml version=1.0 encoding="UTF-8" standalone="no" ?>
<!-- This document describes one book -->
<!DOCTYPE book SYSTEM "book.dtd">
```

- a body with a single root element

```
<book>
  <title>Long live XML</title>
</book>
```

- An optional *epilog* – seldom used

39



Well-formedness Rules

- An XML document must follow “Document” *production*, i.e., document contains a *prolog*, a root element and a miscellaneous part to which the following rules apply
- One root element containing all other elements
- Elements must have both a start and end tag, except that *empty* elements end in “/>”
- Elements must nest, i.e., elements do not overlap, e.g.:
  <section><sectionname> …</sectionname> … </section>
- Attribute values in double or single quotes, e.g.:
  <book settext=“no”>Castro E., XML for the World Wide Web, Peachpit Press, 2000.</book>
- Markup characters (eg, < > &) do not appear in parsed content, use entity reference instead, e.g.: <eqn>1 &lt; 3</eqn> for showing 1 < 3
- Element names may begin with letter or underscore or “:”and remaining characters include alphanumeric, “_”, “-”, “:” and “.”
- Cannot use the same name for two or more attributes of the same element

40

XML and Ajax

- An XHR object can be used to communicate text structured as XML from the server, through the responseXML property of the object
- As long as the TEXT sent from the server is well-formed XML, it will be accessible from responseXML property. Note that responseXML is a DOM document object.
- XML written by the server to an XHR object in this way, may be accessed, queried and manipulated using DOM commands in JavaScript
- XML can also be manipulated on the server using DOM commands (eg in PHP)
- A given application must make decisions about whether to do processing on the server or the client

41

Requesting XML - Example

- We provide an XML file, to reside on the server
- We access this using an XHR object, so that the contents are read into the responseXML property
- We then copy this to a JavaScript variable, and manipulate it a little before displaying in the browser

42

classes.xml

```
<?xml version="1.0"?>
<classes>
   <class>
      <classID>CS115</classID>
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Adams</instructor><title>Programming Concepts</title>
   </class>
   <class>
      <classID semester="fall">CS205</classID>
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Dykes</instructor><title>JavaScript</title>
   </class>
   <class>
      <classID semester="fall">CS255</classID>
      <department>ComputerScience</department><credits req=“no">3</credits>
      <instructor>Brunner</instructor><title>Java</title>
   </class>
</classes>
```

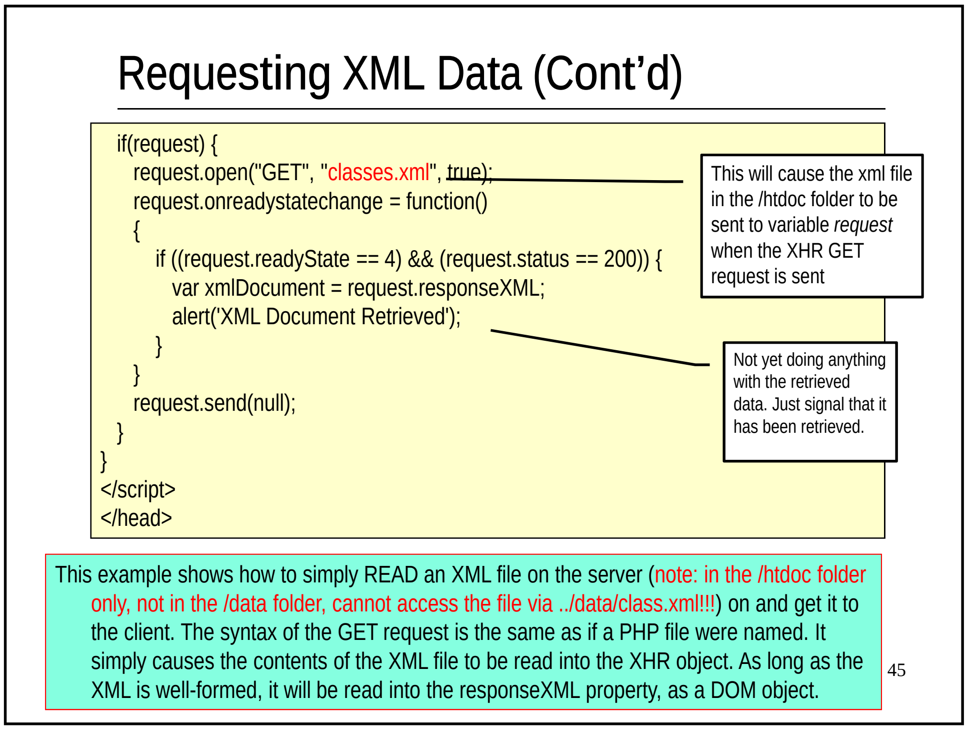
43

Requesting XML Data – Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Requesting XML</title>
<script type="text/javascript" src=“xhr.js"></script>
<script language = "javascript">
function getDoc()
{
    request = createRequest();
    if (request.overrideMimeType) {
        request.overrideMimeType("text/xml");
    }
```

To ensure that responses sent as XML are properly handled

44



Requesting XML Data (Cont'd)

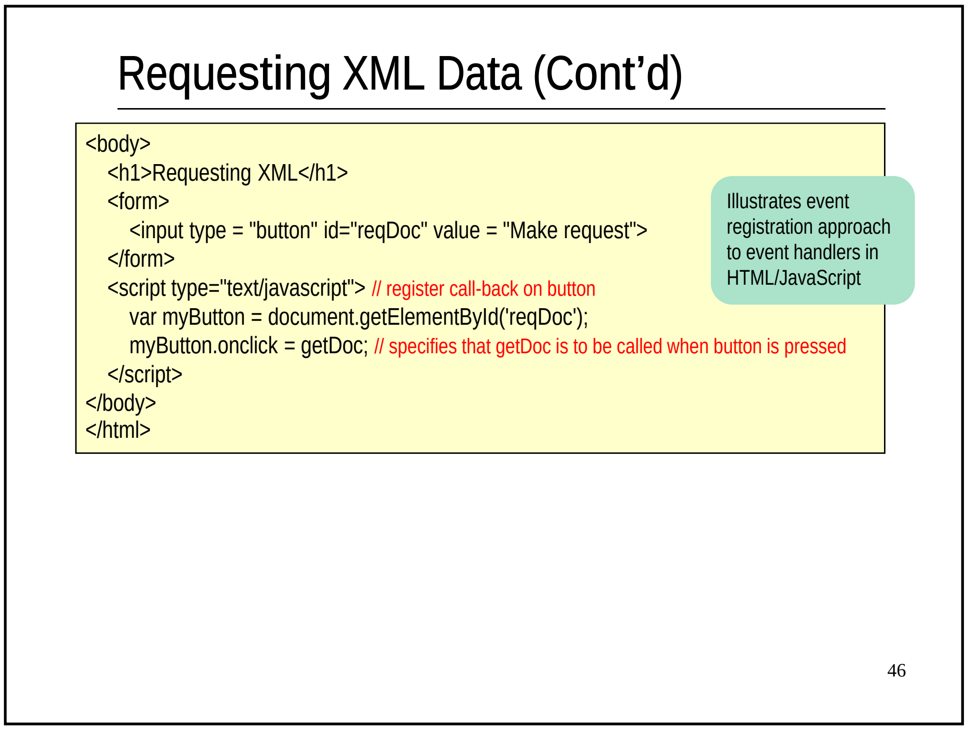
```
  if(request) {
    request.open("GET", "classes.xml", true);
    request.onreadystatechange = function()
    {
       if ((request.readyState == 4) && (request.status == 200)) {
         var xmlDocument = request.responseXML;
         alert('XML Document Retrieved');
       }
    }
    request.send(null);
  }
}
</script>
</head>
```

This will cause the xml file in the /htdoc folder to be sent to variable *request* when the XHR GET request is sent

Not yet doing anything with the retrieved data. Just signal that it has been retrieved.

This example shows how to simply READ an XML file on the server (note: in the /htdoc folder only, not in the /data folder, cannot access the file via ../data/class.xml!!!) on and get it to the client. The syntax of the GET request is the same as if a PHP file were named. It simply causes the contents of the XML file to be read into the XHR object. As long as the XML is well-formed, it will be read into the responseXML property, as a DOM object.

45

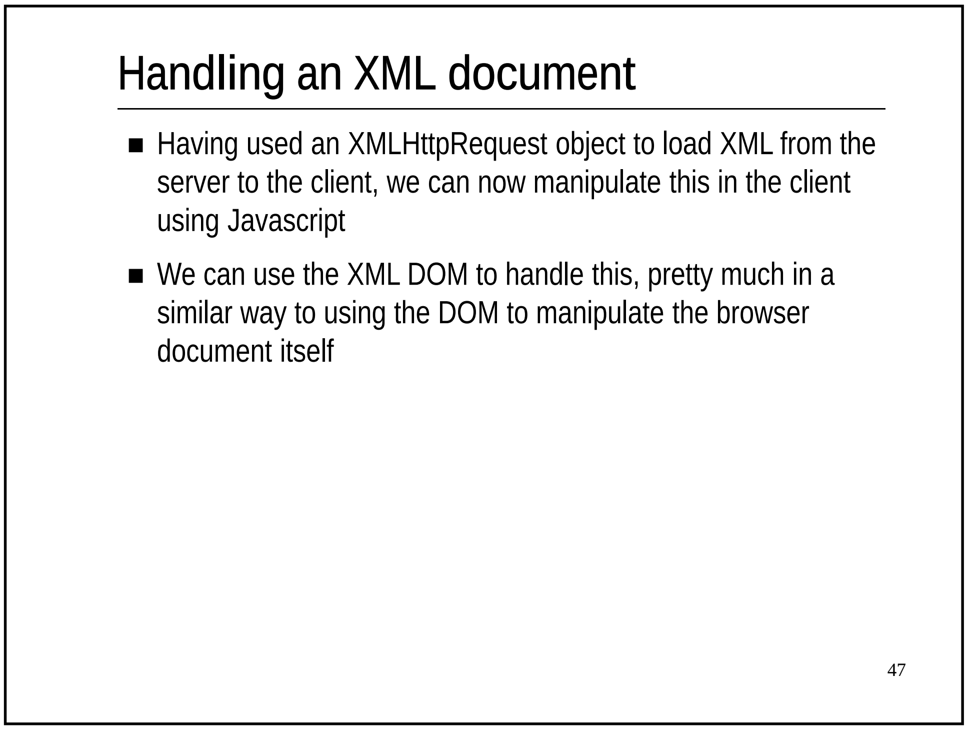


Requesting XML Data (Cont'd)

```
<body>
   <h1>Requesting XML</h1>
   <form>
      <input type = "button" id="reqDoc" value = "Make request">
   </form>
   <script type="text/javascript"> // register call-back on button
      var myButton = document.getElementById('reqDoc');
      myButton.onclick = getDoc; // specifies that getDoc is to be called when button is pressed
   </script>
</body>
</html>
```

Illustrates event registration approach to event handlers in HTML/JavaScript

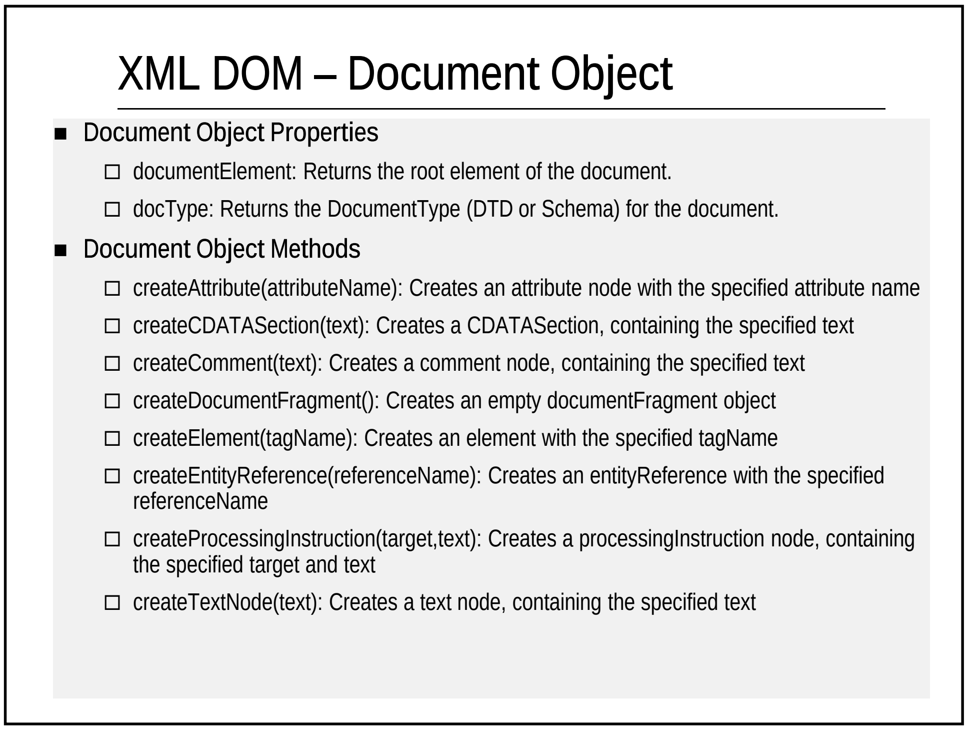
46



Handling an XML document

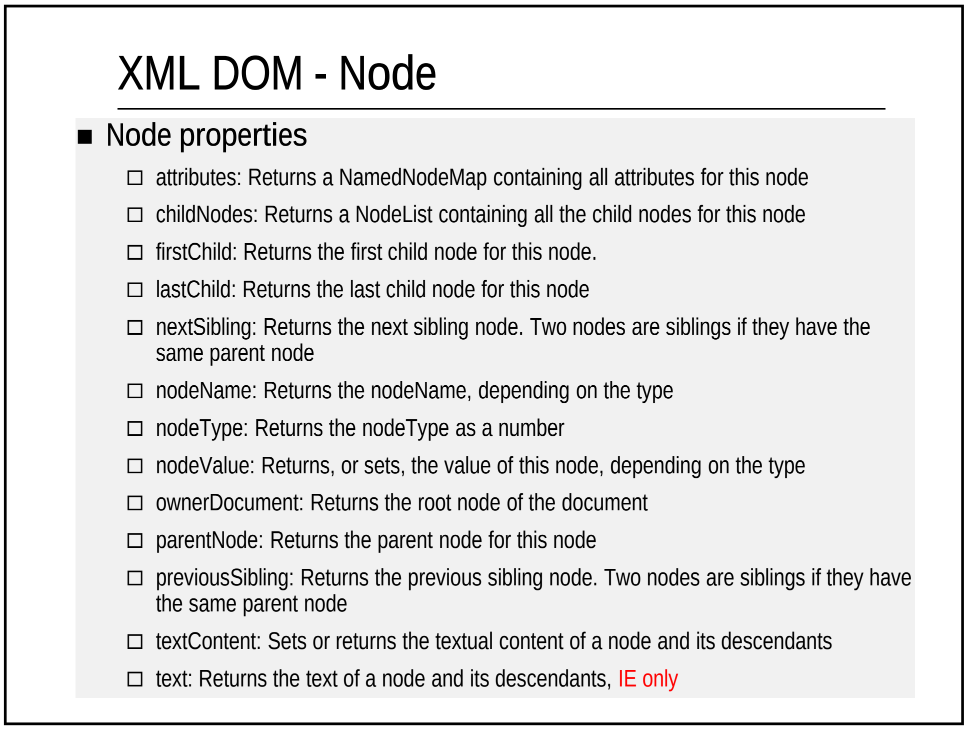
- Having used an XMLHttpRequest object to load XML from the server to the client, we can now manipulate this in the client using Javascript
- We can use the XML DOM to handle this, pretty much in a similar way to using the DOM to manipulate the browser document itself

47



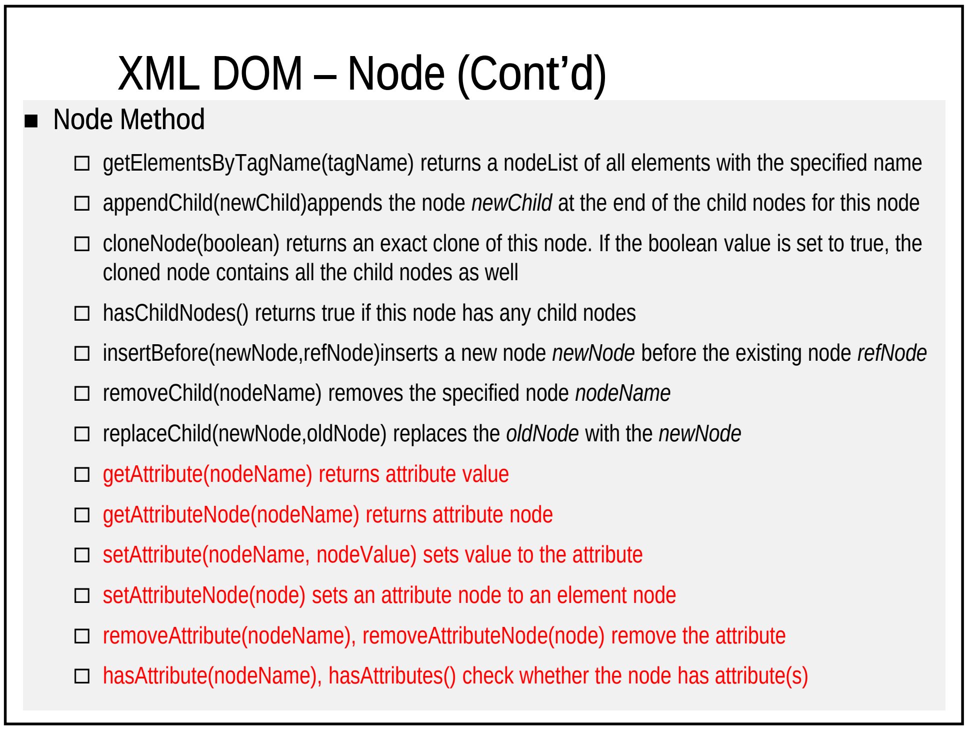
XML DOM – Document Object

- Document Object Properties
  - documentElement: Returns the root element of the document.
  - docType: Returns the DocumentType (DTD or Schema) for the document.
- Document Object Methods
  - createAttribute(attributeName): Creates an attribute node with the specified attribute name
  - createCDATASection(text): Creates a CDATASection, containing the specified text
  - createComment(text): Creates a comment node, containing the specified text
  - createDocumentFragment(): Creates an empty documentFragment object
  - createElement(tagName): Creates an element with the specified tagName
  - createEntityReference(referenceName): Creates an entityReference with the specified referenceName
  - createProcessingInstruction(target,text): Creates a processingInstruction node, containing the specified target and text
  - createTextNode(text): Creates a text node, containing the specified text



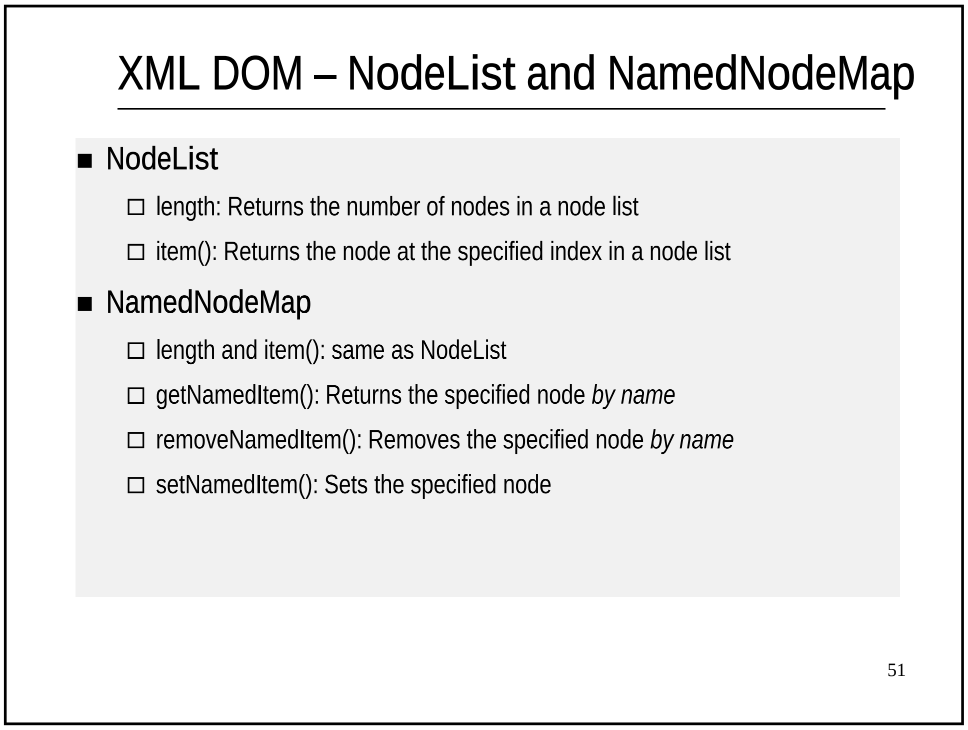
XML DOM - Node

- Node properties
  - attributes: Returns a NamedNodeMap containing all attributes for this node
  - childNodes: Returns a NodeList containing all the child nodes for this node
  - firstChild: Returns the first child node for this node.
  - lastChild: Returns the last child node for this node
  - nextSibling: Returns the next sibling node. Two nodes are siblings if they have the same parent node
  - nodeName: Returns the nodeName, depending on the type
  - nodeType: Returns the nodeType as a number
  - nodeValue: Returns, or sets, the value of this node, depending on the type
  - ownerDocument: Returns the root node of the document
  - parentNode: Returns the parent node for this node
  - previousSibling: Returns the previous sibling node. Two nodes are siblings if they have the same parent node
  - textContent: Sets or returns the textual content of a node and its descendants
  - text: Returns the text of a node and its descendants, IE only



XML DOM – Node (Cont’d)

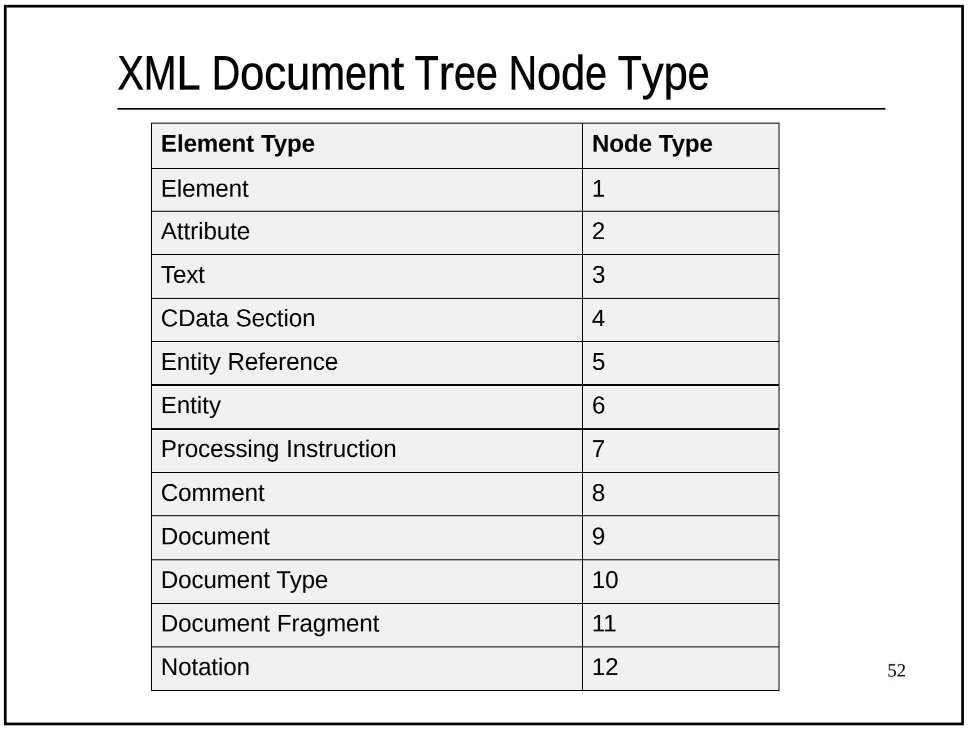
- Node Method
  - getElementsByTagName(tagName) returns a nodeList of all elements with the specified name
  - appendChild(newChild)appends the node *newChild* at the end of the child nodes for this node
  - cloneNode(boolean) returns an exact clone of this node. If the boolean value is set to true, the cloned node contains all the child nodes as well
  - hasChildNodes() returns true if this node has any child nodes
  - insertBefore(newNode,refNode)inserts a new node *newNode* before the existing node *refNode*
  - removeChild(nodeName) removes the specified node *nodeName*
  - replaceChild(newNode,oldNode) replaces the *oldNode* with the *newNode*
  - getAttribute(nodeName) returns attribute value
  - getAttributeNode(nodeName) returns attribute node
  - setAttribute(nodeName, nodeValue) sets value to the attribute
  - setAttributeNode(node) sets an attribute node to an element node
  - removeAttribute(nodeName), removeAttributeNode(node) remove the attribute
  - hasAttribute(nodeName), hasAttributes() check whether the node has attribute(s)



XML DOM – NodeList and NamedNodeMap

- NodeList
  - length: Returns the number of nodes in a node list
  - item(): Returns the node at the specified index in a node list
- NamedNodeMap
  - length and item(): same as NodeList
  - getNamedItem(): Returns the specified node *by name*
  - removeNamedItem(): Removes the specified node *by name*
  - setNamedItem(): Sets the specified node

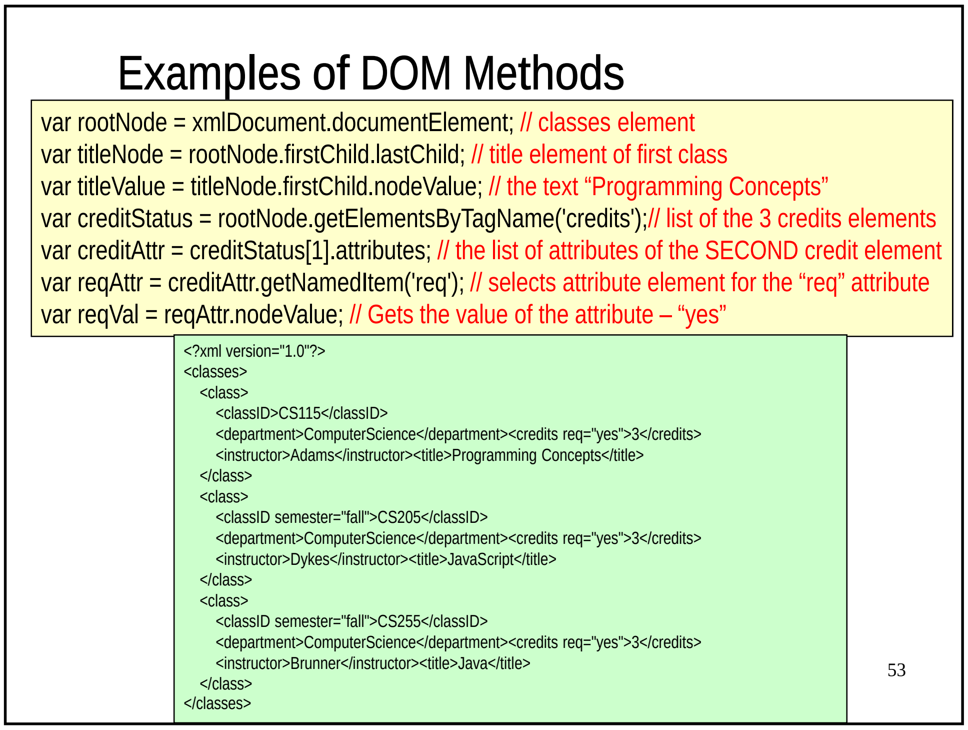
51



XML Document Tree Node Type

Element Type	Node Type
Element	1
Attribute	2
Text	3
CData Section	4
Entity Reference	5
Entity	6
Processing Instruction	7
Comment	8
Document	9
Document Type	10
Document Fragment	11
Notation	12

52

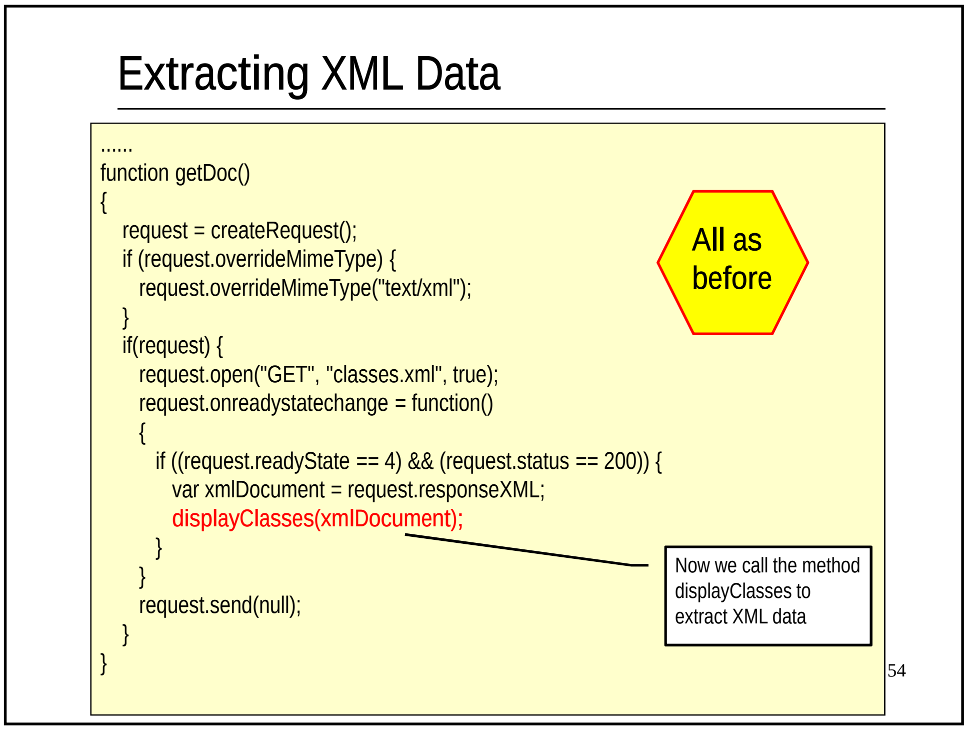


Examples of DOM Methods

```
var rootNode = xmlDocument.documentElement; // classes element
var titleNode = rootNode.firstChild.lastChild; // title element of first class
var titleValue = titleNode.firstChild.nodeValue; // the text “Programming Concepts”
var creditStatus = rootNode.getElementsByTagName('credits');// list of the 3 credits elements
var creditAttr = creditStatus[1].attributes; // the list of attributes of the SECOND credit element
var reqAttr = creditAttr.getNamedItem('req'); // selects attribute element for the “req” attribute
var reqVal = reqAttr.nodeValue; // Gets the value of the attribute – “yes”
```

```xml
<?xml version="1.0"?>
<classes>
   <class>
      <classID>CS115</classID>
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Adams</instructor><title>Programming Concepts</title>
   </class>
   <class>
      <classID semester="fall">CS205</classID>
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Dykes</instructor><title>JavaScript</title>
   </class>
   <class>
      <classID semester="fall">CS255</classID>
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Brunner</instructor><title>Java</title>
   </class>
</classes>
```

53



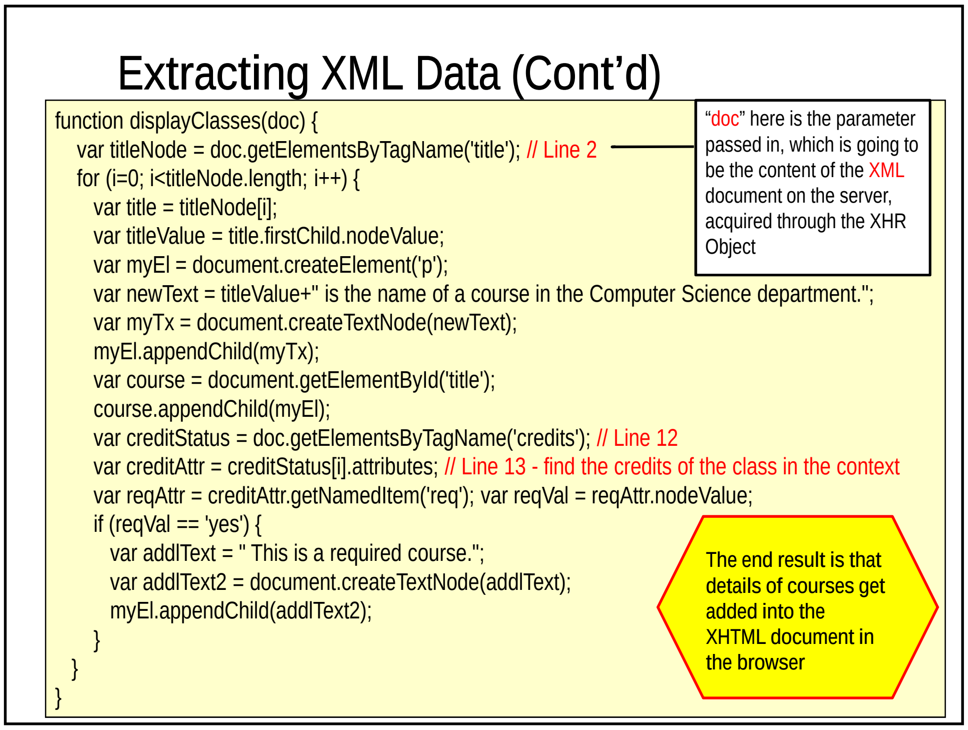
Extracting XML Data

```
......
function getDoc()
{
    request = createRequest();
    if (request.overrideMimeType) {
       request.overrideMimeType("text/xml");
    }
    if(request) {
       request.open("GET", "classes.xml", true);
       request.onreadystatechange = function()
       {
          if ((request.readyState == 4) && (request.status == 200)) {
             var xmlDocument = request.responseXML;
             displayClasses(xmlDocument);
          }
       }
       request.send(null);
    }
}
```

All as before

Now we call the method displayClasses to extract XML data

54

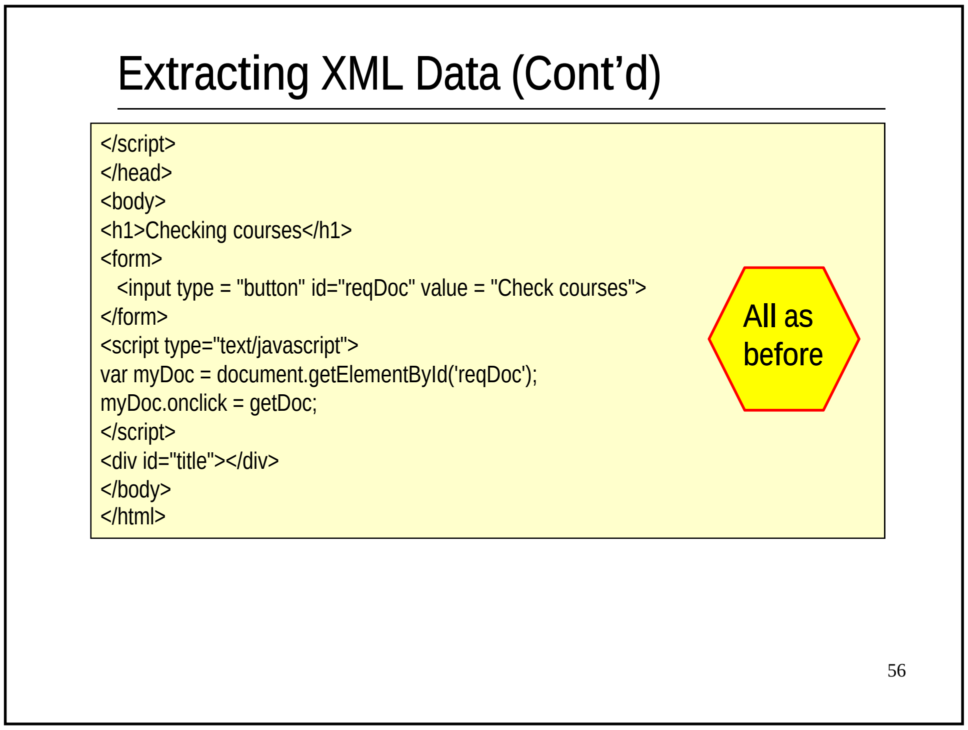


Extracting XML Data (Cont'd)

```
function displayClasses(doc) {
   var titleNode = doc.getElementsByTagName('title'); // Line 2
   for (i=0; i<titleNode.length; i++) {
      var title = titleNode[i];
      var titleValue = title.firstChild.nodeValue;
      var myEl = document.createElement('p');
      var newText = titleValue+" is the name of a course in the Computer Science department.";
      var myTx = document.createTextNode(newText);
      myEl.appendChild(myTx);
      var course = document.getElementById('title');
      course.appendChild(myEl);
      var creditStatus = doc.getElementsByTagName('credits'); // Line 12
      var creditAttr = creditStatus[i].attributes; // Line 13 - find the credits of the class in the context
      var reqAttr = creditAttr.getNamedItem('req'); var reqVal = reqAttr.nodeValue;
      if (reqVal == 'yes') {
         var addlText = " This is a required course.";
         var addlText2 = document.createTextNode(addlText);
         myEl.appendChild(addlText2);
      }
   }
}
```

"doc" here is the parameter passed in, which is going to be the content of the XML document on the server, acquired through the XHR Object

The end result is that details of courses get added into the XHTML document in the browser



Extracting XML Data (Cont'd)

```
</script>
</head>
<body>
<h1>Checking courses</h1>
<form>
   <input type = "button" id="reqDoc" value = "Check courses">
</form>
<script type="text/javascript">
var myDoc = document.getElementById('reqDoc');
myDoc.onclick = getDoc;
</script>
<div id="title"></div>
</body>
</html>
```

All as before

56

Discussion

- Codes in Line 12/Line 13 of Slide 55 do not look good! How to improve them?
  → Your exercise
  (hint: change 'title' in line 2 to 'class')
- How to make the program more general for displaying all information of a document with several levels?
  → See slide 61 and try to understand the *recursive function* by yourself

57

classes2.xml

```
<?xml version="1.0"?>
<classes>
 <class>
   <classID>CS115</classID><department>ComputerScience</department>
   <instructors><instructor>Adams</instructor><instructor>Dykes</instructor></instructors>
   <title>Programming Concepts</title><credits req="yes">3</credits>
  </class>
  <class>
   <classID semester="fall">CS205</classID><department>ComputerScience</department>
   <instructors><instructor>Dykes</instructor><instructor>Doug</instructor></instructors>
   <title>JavaScript</title><credits req="yes">3</credits>
  </class>
  <class>
   <classID semester="fall">CS255</classID><department>ComputerScience</department>
   <instructors><instructor>Chengfei</instructor><instructor>Doug</instructor></instructors>
   <title>Java</title><credits req="no">3</credits>
  </class>
</classes>
```

58

Show Whole XML Tree

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Checking Courses</title>
<script type="text/javascript" src="xhr.js"></script>
<script language = "javascript">
var classInfo = ""; // hold display data
function getDoc()
{
   request = createRequest();
   if (request.overrideMimeType) {
      request.overrideMimeType("text/xml");
   }
```

59

Show Whole XML Tree (Cont'd)

```
   if(request) {
      request.open("GET", "classes.xml", true);
      request.onreadystatechange = function()
      {
         if ((request.readyState == 4) && (request.status == 200)) {
            var xmlDocument = request.responseXML;
            root = xmlDocument.documentElement;
            getInfo(root);
            var display = document.getElementById('info');
            display.innerHTML = classInfo;
         }
      }
      request.send(null);
   }
}
```

Now we call a recursive function to get all class info in the classInfo variable

60

Show Whole XML Tree (Cont’d)

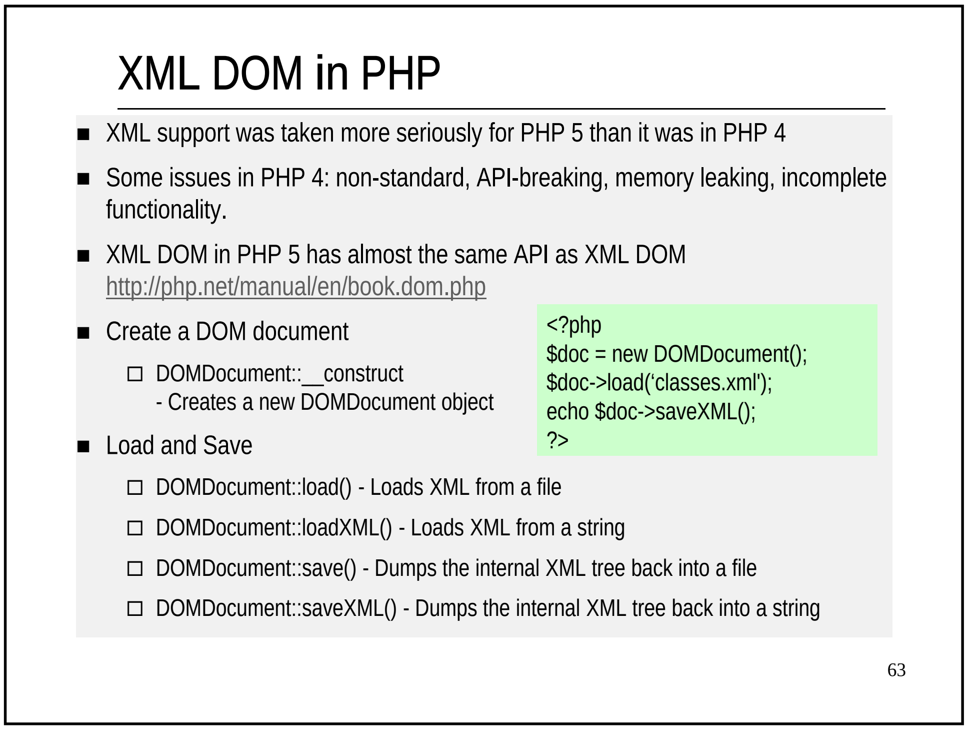
```
function getInfo(n)
{  var i, indent;
   indent = "2em";
   if (n.nodeType == 3) classInfo += "<SPAN>" + n.nodeValue + "</SPAN>";
   else if (n.nodeType == 1) {
        classInfo += "<DIV Style='margin-left:" + indent + " '>" +
        "<b>" + n.nodeName+ "</b>" + getAttributes(n) + "<br>";
        for( i = 0; i < n.childNodes.length; i++ )
            getInfo(n.childNodes.item(i));
        classInfo +="</DIV>";
   }
}

function getAttributes(el)
{  var i;
   var strResult = "";
   xmapAtts = el.attributes;
   for (i = 0; i<xmapAtts.length; i++)
      strResult += ' ' + xmapAtts.item(i).nodeName + '="' + xmapAtts.item(i).nodeValue + '"';
   return(strResult);
}
```

Show Whole XML Tree (Cont’d)

```
</script>
</head>
<body>
<h1>Checking courses</h1>
<form>
  <input type = "button" id="reqDoc" value = "Check courses">
</form>
<script type="text/javascript">
var myDoc = document.getElementById('reqDoc');
myDoc.onclick = getDoc;
</script>
<div id=“info”></div>
</body>
</html>
```

All as before

62

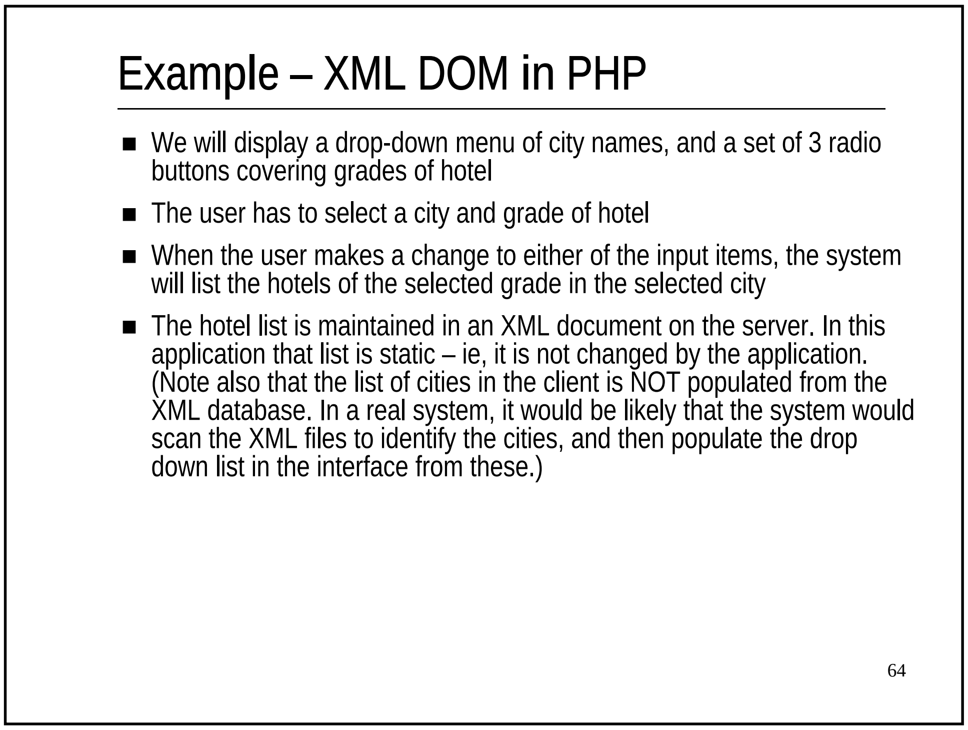


XML DOM in PHP

- XML support was taken more seriously for PHP 5 than it was in PHP 4
- Some issues in PHP 4: non-standard, API-breaking, memory leaking, incomplete functionality.
- XML DOM in PHP 5 has almost the same API as XML DOM http://php.net/manual/en/book.dom.php
- Create a DOM document
  - DOMDocument::__construct
    - Creates a new DOMDocument object
- Load and Save
  - DOMDocument::load() - Loads XML from a file
  - DOMDocument::loadXML() - Loads XML from a string
  - DOMDocument::save() - Dumps the internal XML tree back into a file
  - DOMDocument::saveXML() - Dumps the internal XML tree back into a string

```
<?php
$doc = new DOMDocument();
$doc->load(‘classes.xml');
echo $doc->saveXML();
?>
```

63



Example – XML DOM in PHP

- We will display a drop-down menu of city names, and a set of 3 radio buttons covering grades of hotel
- The user has to select a city and grade of hotel
- When the user makes a change to either of the input items, the system will list the hotels of the selected grade in the selected city
- The hotel list is maintained in an XML document on the server. In this application that list is static – ie, it is not changed by the application. (Note also that the list of cities in the client is NOT populated from the XML database. In a real system, it would be likely that the system would scan the XML files to identify the cities, and then populate the drop down list in the interface from these.)

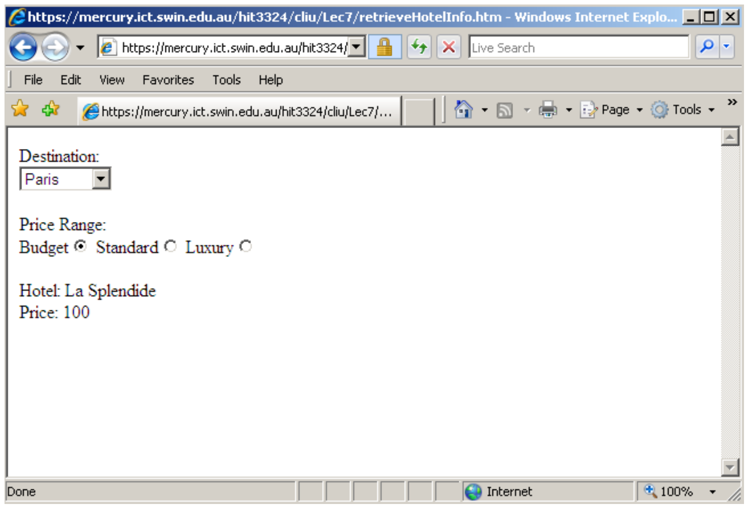
64

hotels.xml

```
<?xml version="1.0"?>
<hotels>
    <hotel>
        <City>Paris</City>
        <Name>La Splendide</Name>
        <Type>Budget</Type>
        <Price>100</Price>
    </hotel>
    <hotel>
        <City>London</City>
        <Name>The Rilton</Name>
        <Type>Luxury</Type>
        <Price>300</Price>
    </hotel>
</hotels>
```

65

Retrieving Hotel Information


https://mercury.ict.swin.edu.au/hit3324/cliu/Lec7/retrieveHotelInfo.htm - Windows Internet Explo...
File Edit View Favorites Tools Help
Live Search
Page Tools
Done
Internet
100%

Destination:
Paris

Price Range:
Budget [✓ ●] Standard ○ Luxury ○

Hotel: La Splendide
Price: 100

66

retrieveHotelInfo.htm

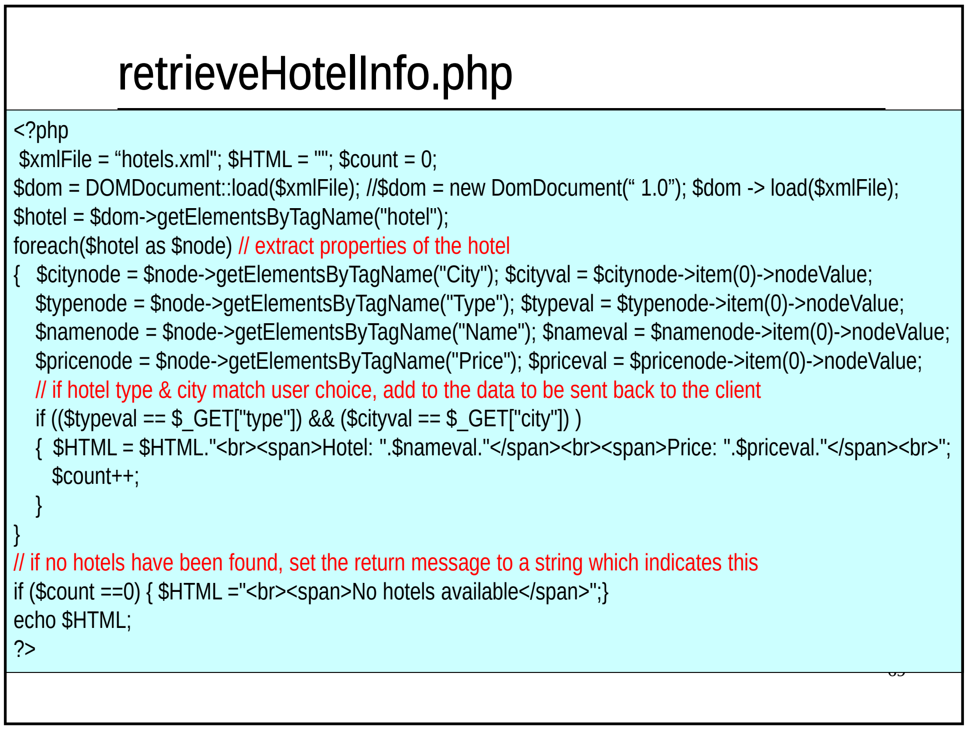
```
<html>
<head><script type="text/javascript" src=“retrieveHotelInfo.js"></script> </head>
<body>Destination:<br />
<select id="selectCity" onchange="retrieveInformation()">
        <option value="London" selected="true">London</option>
        <option value="Paris">Paris</option>
        <option value="New York">New York</option>
        <option value="Chicago">Chicago</option>
        <option value="Seattle">Seattle</option>
</select>
<br /><br />Price Range:<br />
Budget<input name="range" value="Budget" type="radio"  onclick="retrieveInformation()"/>
Standard<input name="range" value="Standard" type="radio" onclick="retrieveInformation()"
checked="true"/>
Luxury<input  name="range" value="Luxury" type="radio"  onclick="retrieveInformation()"/>
<div id="information"></div>
</body>
</html>
```

67

retrieveHotelInfo.js

```
var xHRObject = false;
if (window.XMLHttpRequest) xHRObject = new XMLHttpRequest();
else if (window.ActiveXObject) xHRObject = new ActiveXObject("Microsoft.XMLHTTP");
function retrieveInformation()
{  var city = document.getElementById("selectCity").value;
   var type = "";
   var input = document.getElementsByTagName("input");
   for (var i=0; i < input.length; i++) {
       if (input.item(i).checked == true) type = input.item(i).value;
   }
   xHROobject.open("GET", "retrieveHotelInfo.php?id=" + Number(new Date) +"&city=" + city +
"&type=" + type, true);
   xHRObject.onreadystatechange = function() {
        if (xHRObject.readyState == 4 && xHRObject.status == 200)
            document.getElementById('information').innerHTML = xHRObject.responseText;
   }
   xHRObject.send(null);
}
```

68



retrieveHotelInfo.php

```
<?php
 $xmlFile = “hotels.xml"; $HTML = ""; $count = 0;
$dom = DOMDocument::load($xmlFile); //$dom = new DomDocument(“ 1.0”); $dom -> load($xmlFile);
$hotel = $dom->getElementsByTagName("hotel");
foreach($hotel as $node) // extract properties of the hotel
{   $citynode = $node->getElementsByTagName("City"); $cityval = $citynode->item(0)->nodeValue;
    $typenode = $node->getElementsByTagName("Type"); $typeval = $typenode->item(0)->nodeValue;
    $namenode = $node->getElementsByTagName("Name"); $nameval = $namenode->item(0)->nodeValue;
    $pricenode = $node->getElementsByTagName("Price"); $priceval = $pricenode->item(0)->nodeValue;
    // if hotel type & city match user choice, add to the data to be sent back to the client
    if (($typeval == $_GET["type"]) && ($cityval == $_GET["city"]) )
    {  $HTML = $HTML."<br><span>Hotel: ".$nameval."</span><br><span>Price: ".$priceval."</span><br>";
       $count++;
    }
}
// if no hotels have been found, set the return message to a string which indicates this
if ($count ==0) { $HTML ="<br><span>No hotels available</span>";}
echo $HTML;
?>
```