Web Application Development: Ajax and Server-Side Technologies and Managing State Information

Week 7

Content

- Ajax technology for client/server communication
- Using XML with XHR objects
- Using cookies to save state information
- Using sessions to save state information
- Shopping cart example

simpleajax.js (in Lecture 1) – object: xhr.

```
var xhr = new XMLHttpRequest();
xhr.open("GET", url, true);
xhr.onreadystatechange = function() {
   if (xhr.readyState == 4 && xhr.status == 200) {
                  place.innerHTML = xhr.responseText;
              } // end if
      } // end anonymous call-back function
xhr.send(null);
```

Step 1: Submitting Data to the Server

- Three steps:
  1. Call the *open* method on the XHR object with the request to establish the client-server link
  2. Specify the call-back function that will execute on the client when the client gets notification that the state of the XHR object has changed
     - *Guarded by a conditional that ensures processing only occurs when the state has been changed to "loaded"*
  3. Send the request, to activate the communication
- Note that the client keeps being open for further user-interaction whilst the server processes the request (which may take some time to complete)

Submitting Data to the Server (cont'd)

- Assume **xhr** has been defined as an XMLHttpRequest object
- *open* method

```
xhr.open(method, URL to call, asyn or syn)
```

"true" means asynchronous; "false" means synchronous

- Two request methods
  - HTTP *Get*

```
xhr.open("GET", "response.php?value="+data, true);
xhr.send(null);
```

  - HTTP *Post*

```
var argument = "value="; argument += encodeURIComponent(data);
xhr.open("POST", "response.php", true);
xhr.send(argument);
```

More on POST next week!

Step 2: The Server Receives the Request

- The request sent by **GET** or **POST** is part of either the **URL** (in the case of GET) or the ***Request* body** (in the case of POST)
- The server picks up these items as variables
  - PHP uses $_*GET*, $_*POST*, or $_*REQUEST*

Client

```
<input type="text" id="MyTextbox1" name="MyTextbox1" />
```

Server

```
$Tb = $_GET["MyTextbox1"];

$Tb = $_POST["MyTextbox1"];

$Tb = $_REQUEST["MyTextbox1"];
```

Essentially the parameters are transmitted as an associative array, with parameter names and values associated together. Hence they can be picked up on the server as indicated in the above example.

XML and Ajax

- The benefit of XML with Ajax is that structured data may be
  - assembled or stored in XML form on the server, use being made of the DOM API or other APIs
  - processed using a server-side language on the server (eg, PHP DOM API, XSLT)
  - sent from server to client (to the responseXML property of an XHR object)
  - processed using JavaScript on the client (using the DOM API)

Well-formed and Valid XML

- XML must be well-formed, i.e., it must conform to all well-formedness rules (*compulsory*).
- XML is valid with respect to a DTD if it satisfies the requirements of the DTD (*optional*).

DTD (Document Type Definition)

XML for Several Persons

```
<?xml version="1.0" encoding="UTF-8"?>
<Persons>
    <Person>
        <Name>
            <First>Thomas</First>
            <Last>Atkins</Last>
        </Name>
        <Age>30</Age>
    </Person>
    <Person>
        <Name>
            <First>Sachin</First>
            <Last>Tendulkar</Last>
        </Name>
        <Age>38</Age>
    </Person>
</Persons>
```

A DTD for the XML

Persons.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Persons ((Person+))>
<!ELEMENT Person ((Name, Age))>
<!ELEMENT Name ((First, Last))>
<!ELEMENT Last (#PCDATA)>
<!ELEMENT First (#PCDATA)>
<!ELEMENT Age (#PCDATA)>
```

XML for Persons with a DTD

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Persons SYSTEM “Persons.dtd">
<Persons>
    <Person>
        <Name>
            <First>Thomas</First>
            <Last>Atkins</Last>
        </Name>
        <Age>30</Age>
    </Person>
    <Person>
        <Name>
            <First>Sachin</First>
            <Last>Tendulkar</Last>
        </Name>
        <Age>38</Age>
    </Person>
</Persons>
```

XML on the Server

- New XML documents (see example)
  - Create an XML DOM object (construct method)
  - Then add various nodes by XML DOM API
- Existing XML documents
  - Create an XML DOM object (construct method)
  - Load an XML document stored on the server (load method)
  - Read or update nodes by XML DOM API
- Store XML documents
  - Save an XML document on the server (save method)
  - "Serialise" an XML document as a string (saveXML method)

XML on the Server

- Send XML documents to the client
  - Just "echo" the text form of an XML document, which loads the document to the XHR object on the client
- We illustrate the manipulation of XML using PHP on the server by a simple example (next slide)
- If this PHP file was "called" in an Ajax application, we can access the returned XML document using responseXML property of the XHR object
- If we just "run" the PHP "program" directly on the server using "php XML.php", the XML text will be printed to the screen

Step 3: Constructing/Writing the HTTP Response in XML

File XML.php

```
<?
$doc = new DomDocument('1.0');
$root = $doc->createElement('ajax');
$doc->appendChild($root);
$child = $doc->createElement('js');
$root->appendChild($child);
$value = $doc->createTextNode("coordination");
$child->appendChild($value );
$strXml = $doc->saveXML();
echo $strXml
?>
```

XML created

```
<?xml version="1.0"?>
<ajax>
    <js>coordination</js>
</ajax>
```

- $doc is a PHP variable that represents a structured XML document
- The saveXML() method returns the string representation of that object
- echo expects a string as its argument

Notes on the Some of the Code

```
<?
1:$doc = new DomDocument('1.0');
2:$root = $doc->createElement('ajax');
3:$doc->appendChild($root);
```

- 1: creates a new Dom document and assigns it to variable $doc
- 2: creates a new element and assigns it to the variable $root
  - Note that createElement is a method of the DomDocument class in PHP, and it creates a new element for the DomDocument object.
- 3: appends $root as a child of the $doc element. (Note that this method call does return a value – namely the node, $root, that was appended to $doc, but we just ignore the return value and treat the method as if it had been declared as a void method.)

Step 4: Receiving Data from the Server

- The data sent from the server will be received by the XHR object.
- To access this for processing on the client, we have to set up a callback function in the client-side JavaScript program.
- The function should check *readyState* of the XHR object to see if the data load is complete (*readyState* = 4)? If yes, we also check the status of the XHR object to see if the data transmission has been completed without error (eg, status = 200).
- If readyState is not 4, do nothing and wait for the next state change that calls the function again. Else we check the status, and if it is 200, then we proceed to process the received data on the client.
- There are two properties of the XHR object to choose to examine data from server
  - responseText – holds the response as a string
  - responseXML – holds the response as an XML DOM object (which can be manipulated as such in JavaScript)

Receiving Data from the Server (cont'd)

- Assume that method getData has been set as the call-back function on the readyStateChange event on the XHR object; thus it is called every time the readyState property changes its value.
- **We only want the processing to occur when the state has changed to value 4.** The function will be called many times before the desired final state is reached (the state will change from 0 to 1 to 2 to 3 and then to 4). We include an "if" statement that only processes the data when we have assurance that the data is there! (In the *simpleajax* example, we had the call-back function alert the value of xhr.readyState, so that we could see the progression of its value from 0 through to 4 with successive event occurrences.)

Standard pattern in call-back function

```
function getData()
{
  if ((xhr.readyState == 4) && (xhr.status == 200))
  {
    // processing here
  }
}
```

Receiving Data from the Server (cont’d)

- Alternatively we can nest the tests in the condition in the call-back function, so that if the status is not 200, we can signal that this is the issue (generally it means that something has gone wrong):

```
function getData()
{
  if (xhr.readyState == 4)
      if (xhr.status == 200)
      {
        // data received and ok : process it
        // put the code for processing the data here
      }
     else
     {
          alert (“status problem”);
      }
}
```

The responseText Property

- At the client side, use a JavaScript variable to collect the contents of the response from xhr.responseText if it has been sent as a text string

```
// client side
var text = xhr.responseText;
```

- At the server side, use PHP to prepare the data, and send as a text string

```
// PHP at the server side
$data = “This is returned data”;
echo $data;
```

- We can use responseText to retrieve a serialised XML document, but will lose many benefits – it is not in the form of a DOM document, ready to be processed by JavaScript.

The responseXML Property

- At the server side, use either PHP (or another server-side language) to prepare the XML document

```
// PHP at the server side
$data = “<?xml version=\“1.0\” encoding=\“ISO-8859-1\”
standalone=\“yes\”?> <root><child>Data</child></root>”;
header("Content-type: text/xml");   // have to set this for IE
echo $data;
```

- At the client side, use a JavaScript variable to collect the contents of the *response from xhr.responseXML*

```
// client side
var myDoc = xhr.responseXML;
```

```
// client side using Firefox
xhr.overrideMimeType(“text/xml”); // if ContentType is not set at server side
xhr.send(null);                   // assuming GET protocol
……..
var myDoc = xhr.responseXML;
```

Content-type

- At the server side: be in the habit of specifying the Content-type

```
// PHP at the server side
$data = “<?xml version=\“1.0\” encoding=\“ISO-8859-1\”
standalone=\“yes\”?> <root><child>Data</child></root>”;
header("Content-type: text/xml");   // have to set this for IE
echo $data;
```

- At the client side, if YOU did not write the server-side code, and thus can’t be sure if the Content-type was actually set, override the Mime type in the XHR object, before initiating the communication, to be sure Firefox will treat it correctly

```
// client side using Firefox
xhr.overrideMimeType(“text/xml”); // if ContentType is not set at server side
xhr.send(null);                   // assuming GET protocol
……..
var myDoc = xhr.responseXML;
```

Debugging responseXML

If responseXML gets no content :

- Step 1: Check if there is content using responseText

```
var text = xhr.responseText;
alert(text);
```

- Step 2
  - IE: use errorCode for the error code and reason for error message

```
var errorCode = xHRObject.responseXML.parseError.errorCode;

var errorMessage = xHRObject.responseXML.parseError.reason;
```

  - Firefox: use JavaScript Console, or Firebug to detect what may be wrong

Step 5: Fetching Data from responseXML

- Use DOM API in JavaScript to access nodes

```
<?xml version="1.0"?>
<cart>
  <book>
    <Title>Beginning Ajax</Title>
    <Qty>1</Qty>
  </book>
</cart>
```

```
var myCart = xhr.responseXML;
var books =
myCart.getElementsByTagName("book");
var titleNode = books[0].firstChild;
var qtyNode = titleNode.nextSibling;
```

```
// now extract the actual title – in IE (4+)
var title = titleNode.text
```

```
// now extract the actual title in WC3 compliant
// browser  but NOT in IE8!
var title = titleNode.textContent
```

```
// in Safari, have to use
var title = titleNode.innerHTML
```

Server-Side Technologies

- PHP
  - Not tied to Windows
  - Programming language in its own right
  - Run on major web servers: IIS and Apache
- ASP.NET
  - Microsoft's technology, tied to Windows
  - .NET Framework, .NET 2.0
  - A compatible web server, usually IIS
  - C# or Visual Basic.NET
- Java Servlets
  - server-side applications written in Java
  - Used to create interactive web pages
  - Provide similar functions as ASP.NET and PHP

We use PHP exclusively in WAD

Understanding State Information

- Information about individual visits to a Web site is called **state information**
- HTTP was originally designed to be **stateless** – Web browsers store no persistent data about a visit to a Web site
- **Maintaining state** means to store persistent information about Web site visits, that can be passed backwards and forwards between the client and the server.
- Some reasons for maintaining state information
  - Temporarily store information for a user as a browser navigates within a multipart form
  - Provide shopping carts that store order information
  - Use counters to keep track of how many times a user has visited a site
  - Store user IDs and passwords

Using Cookies to Save State Information

- **Cookies** are small pieces of information about a user that are stored by a Web server in text files on the user's computer
- **Temporary cookies** remain available only for the current browser session
- **Persistent cookies** remain available beyond the current browser session and are stored in a text file on a client computer
- Each individual server or domain can store only 20 cookies on a user's computer
- Total cookies per browser cannot exceed 300
- The largest cookie size is 4 kilobytes

Creating Cookies

- To create a cookie, use the `setcookie()` function.

  `setcookie(`*`name`*` [,`*`value`*` ,`*`expires`*`, `*`path`*`, `*`domain`*`, `*`secure`*`])`

- To skip the `value`, `path`, and `domain` parameters, specify an empty string as the parameter value.
- To skip the `expires` and `secure` parameters, specify 0 as the parameter value.
- Call the `setcookie()` function before sending the Web browser any output.

`name` and `value` Parameters

- Cookies created with only the `name` and `value` parameters of the `setcookie()` function are *temporary cookies* because they are available for only the current browser session

```
<?php
setcookie("firstName", "Don");
?>
```

No "expires" parameter for temporary cookies

- The `setcookie()` function can be called multiple times to create additional cookies – as long as the `setcookie()` statements come ***before*** any output on a Web page

```
    setcookie("firstName", "Don");
    setcookie("lastName", "Gosselin");
    setcookie("occupation", "writer");
```

`expires` Parameter

- The `expires` parameter determines how long a cookie can remain on a client system before it is deleted
- Cookies created without an `expires` parameter are available for only the current browser session
- To specify a cookie's expiration time, use PHP's `time()` function

```
setcookie("firstName", "Don", time()+3600);
```

This "expires" parameter, is set to current time + 3600 seconds

`path` Parameter

- The `path` parameter determines the availability of a cookie to other Web pages on a server
- Using the `path` parameter allows cookies to be shared across a server
- A cookie is available to all Web pages in a specified path as well as all subdirectories in the specified path

```
 setcookie("firstName", "Don", time()+3600,
"/marketing/");
```

```
 setcookie("firstName", "Don", time()+3600, "/");
```

`domain` and `secure` Parameters

- The `domain` parameter is used for sharing cookies across multiple servers in the same domain
- Cookies *cannot* be shared outside of a domain

```
setcookie("firstName", "Don", time()+3600, "/",
".gosselin.com");
```

- The `secure` parameter indicates that a cookie can only be transmitted across a secure Internet connection using HTTPS or another security protocol
- To use this parameter, assign a value of 1 (for true) or 0 (for false) as the last parameter of the `setcookie()` function

```
setcookie("firstName","Don",time()+3600,"/",
".gosselin.com", 1);
```

Reading Cookies

- Cookies that are available to the current Web page are automatically assigned to the `$_COOKIE` autoglobal
- Access each cookie by using the cookie name as a key in the associative `$_COOKIE[]` array

```
    echo $_COOKIE['firstName'];
```

- Newly created cookies are *not available* until after the current Web page is reloaded
- To ensure that a cookie is set before you attempt to use it, use the `isset()` function

```
setcookie("firstName", "Don");
setcookie("lastName", "Gosselin");
setcookie("occupation", "writer");
if (isset($_COOKIE['firstName'])&&
 isset($_COOKIE['lastName'])&& isset($_COOKIE['occupation']))
   echo "{$_COOKIE['firstName']}
        {$_COOKIE['lastName']}
        is a {$_COOKIE['occupation']}.";
```

Reading Cookies (continued)

- Can use multidimensional array syntax to set and read cookie values

```
setcookie("professional[0]", "Don");
setcookie("professional[1]", "Gosselin");
setcookie("professional[2]", "writer");
if (isset($_COOKIE['professional']))
     echo "{$_COOKIE['professional'][0]}
          {$_COOKIE['professional'][1]} is a
          {$_COOKIE['professional'][2]}.";
```

Deleting Cookies

- To delete a persistent cookie before the time assigned to the `expires` parameter elapses, assign a new expiration value that is sometime in the past
- Do this by subtracting any number of seconds from the `time()` function

```
setcookie("firstName", "", time()-3600);
setcookie("lastName", "", time()-3600);
setcookie("occupation", "", time()-3600);
```

Using Sessions to Save State Information

- A **session** refers to a period of activity when a PHP script stores *state information on a Web server*
- **Sessions** are a simple way to maintain state information for individual users against a *unique session ID* even when clients disable cookies in their Web browsers. This can be used to persist state information *between page requests*.
- Session IDs are normally sent to the browser via session cookies or passed by URLs.
- The Session ID of current session can be obtained by calling **`session_id()`**
- The absence of an ID or session cookie lets PHP know to create a new session and generate a new session ID.

Starting a Session

- The `session_start()` function starts a new session or continues an existing one. For starting a new session, it generates a unique session ID.
- The `session_start()` function creates a text file on the Web server that is the same name as the session ID, preceded by `sess_`
- Session ID text files are stored in the Web server directory specified by the `session.save_path` directive in your *php.ini* configuration file
- You must call the `session_start()` function ***before*** you send the Web browser any output

Starting a Session (continued)

- If a client's Web browser is configured to accept cookies, the session ID is assigned to a temporary cookie named `PHPSESSID`
- Pass the session ID as a query string to any Web pages that are called as part of the current session

```
<?php
session_start();
...
?>
<p><a href='<?php echo "occupation.php?PHPSESSID="
        . session_id() ?>'>Occupation</a></p>
```

`$_SESSION` autoglobal

- Session state information is stored in the `$_SESSION` autoglobal
- The `$_SESSION` autoglobal can hold several variables
- When the `session_start()` function is called, PHP either initializes a new `$_SESSION` autoglobal or retrieves any variables for the current session (based on the session ID) into the `$_SESSION` autoglobal

Registering/Unregistering Session Variables

```
<?php
session_set_cookie_params(3600);
session_start();
$_SESSION['firstName'] = "Don";
$_SESSION['lastName'] = "Gosselin";
$_SESSION['occupation'] = "writer";
?>
```

Sets the “lifetime” cookie parameter defined in the *php.ini* file to 3600 seconds

```
<?php
session_start();
Unset($_SESSION['occupation']);
?>
```

Using Session Variables

- Use the `isset()` function to ensure that a session variable is set before you attempt to use it

```
<?php
session_start();
if (isset($_SESSION['firstName']) &&
    isset($_SESSION['lastName'])&&
    isset($_SESSION['occupation']))
    echo "<p>" . $_SESSION['firstName'] . " "
          . $_SESSION['lastName'] . " is a "
          . $_SESSION['occupation'] . "</p>";
?>
```

Alternative Way - Registering Session Variables

- Call the `session_register()` function to register one or more global variables.
- If `session_start()` was not called before this function is called, an implicit call to `session_start()` will be made.
- This function has been deprecated as of PHP 5.3 and removed as of PHP 5.4.

Deleting a Session

- Use the `session_destroy()` function to destroy all data registered to a session

```
<?php
session_start();// this has to be called
$_SESSION = array();// unset all session variables
// Delete the session cookie if used
if (ini_get("session.use_cookies")) {
   $params = session_get_cookie_params();
   setcookie(session_name(), '', time() - 42000,
   $params["path"], $params["domain"], $params["secure"] );
}
session_destroy();
?>
```

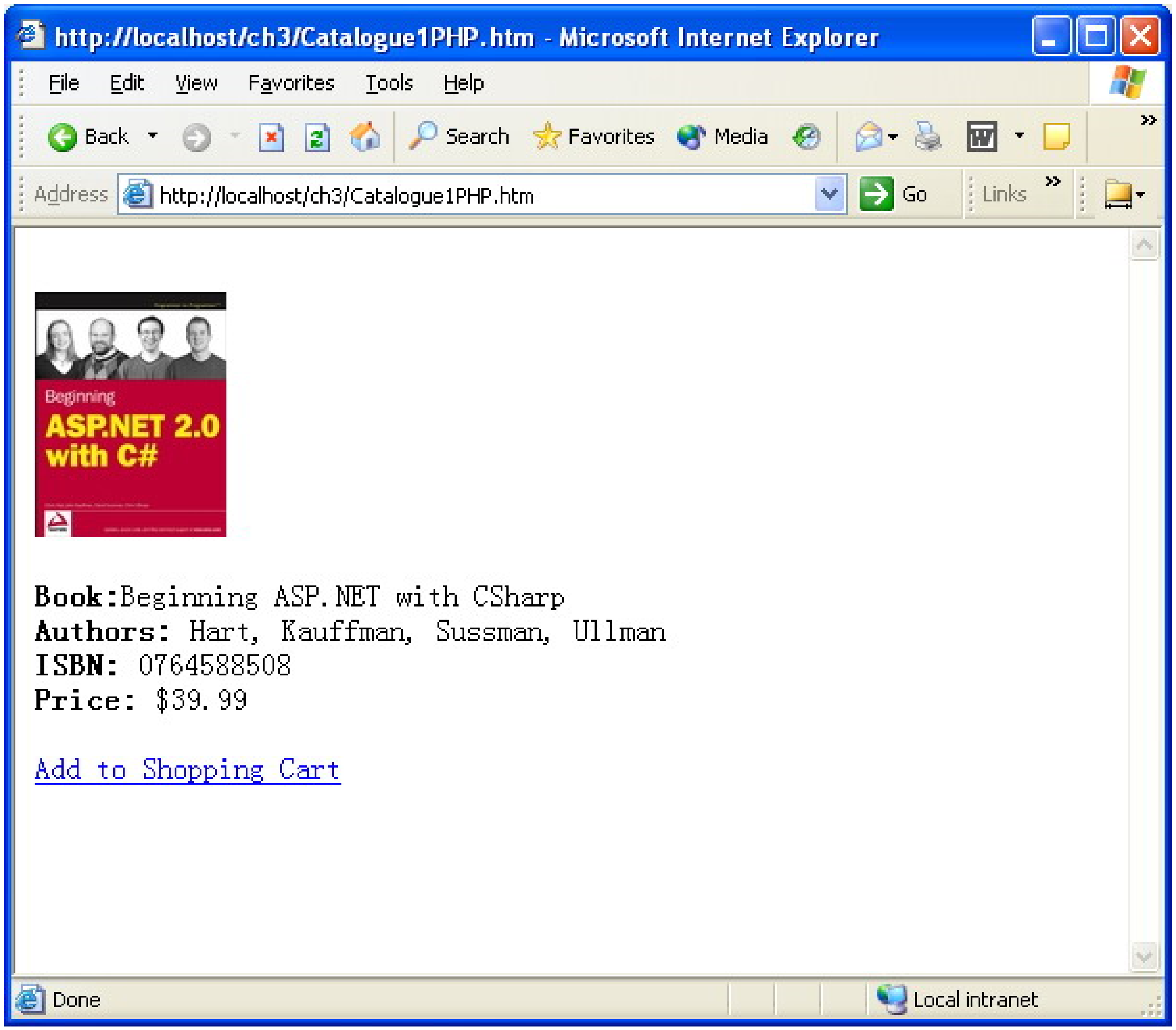
Shopping Cart Example

- Create a dummy catalogue page for a book seller using a shopping cart (the book seller sells just ONE book, for now)
- Allow users to place items in the shopping cart
- Allow users to update the shopping cart without the need to refresh the page (buy additional; remove all)
- Assume the user has been identified
- The shopping cart has three simple functions
  - Can add new items to it
  - The quantity will increase by one if a second item is added
  - Can remove all items from it

Shopping Cart Example

https://mercury.swin.edu.au/wlai/lec7/Catalogue1PHP.htm

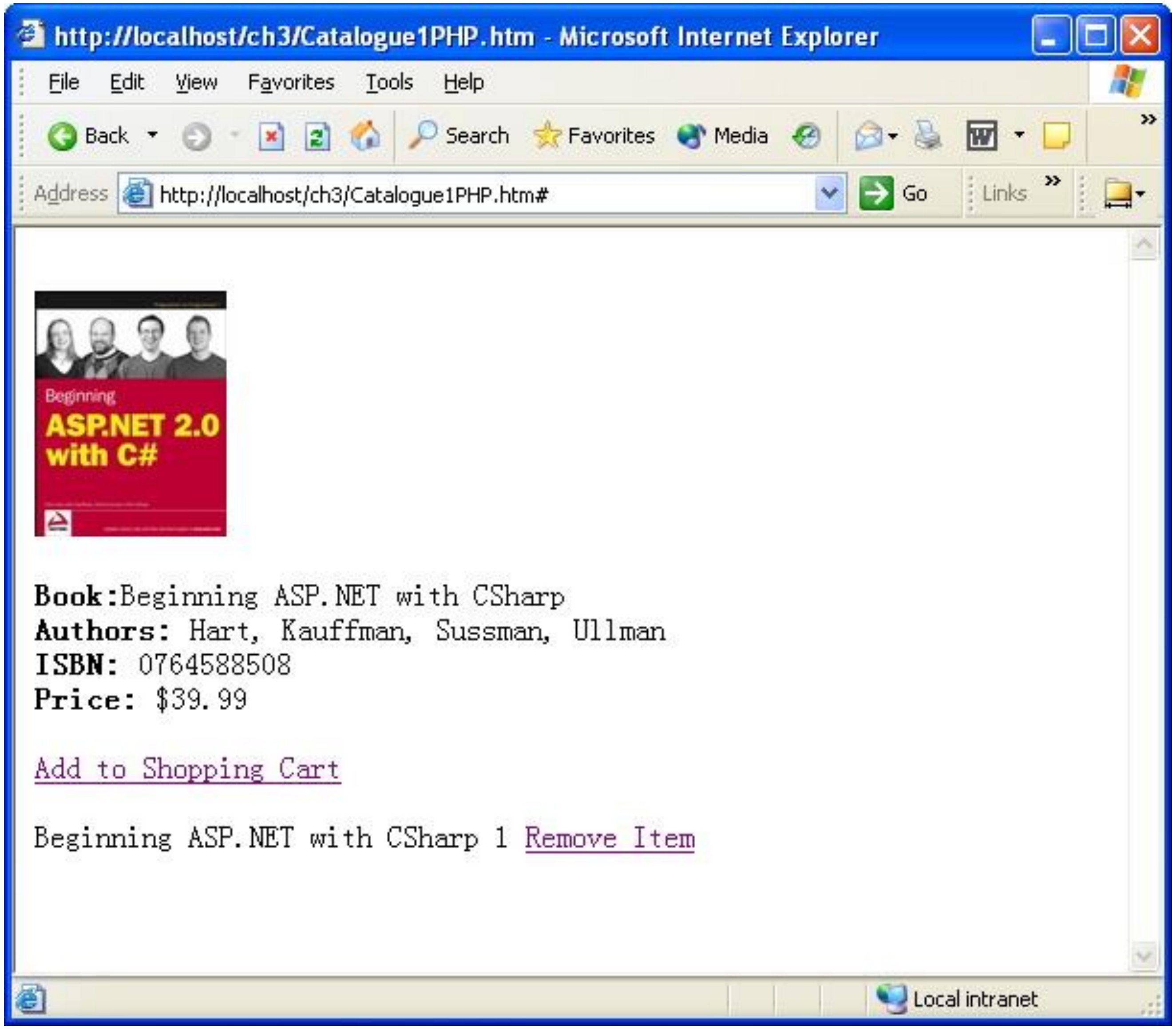
Initial Page


http://localhost/ch3/Catalogue1PHP.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch3/Catalogue1PHP.htm
Go
Links
Beginning ASP.NET 2.0 with C#
Done
Local intranet

Book: Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: $39.99

Add to Shopping Cart

Clicking Add to Shopping Cart

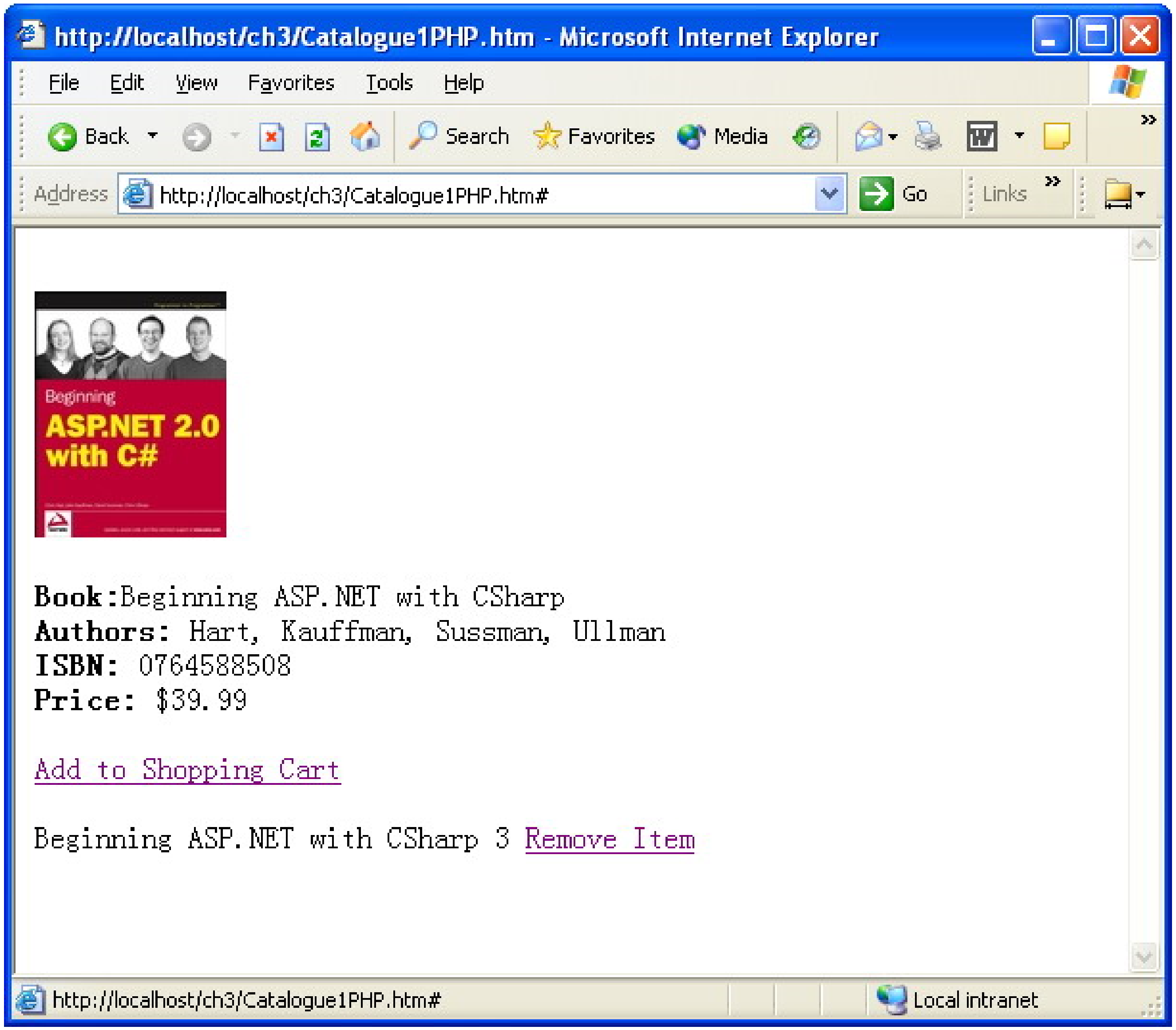

http://localhost/ch3/Catalogue1PHP.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch3/Catalogue1PHP.htm#
Go Links
Beginning ASP.NET 2.0 with C#
Local intranet

Book: Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: $39.99

Add to Shopping Cart

Beginning ASP.NET with CSharp 1 Remove Item

Clicking Add Two More Times

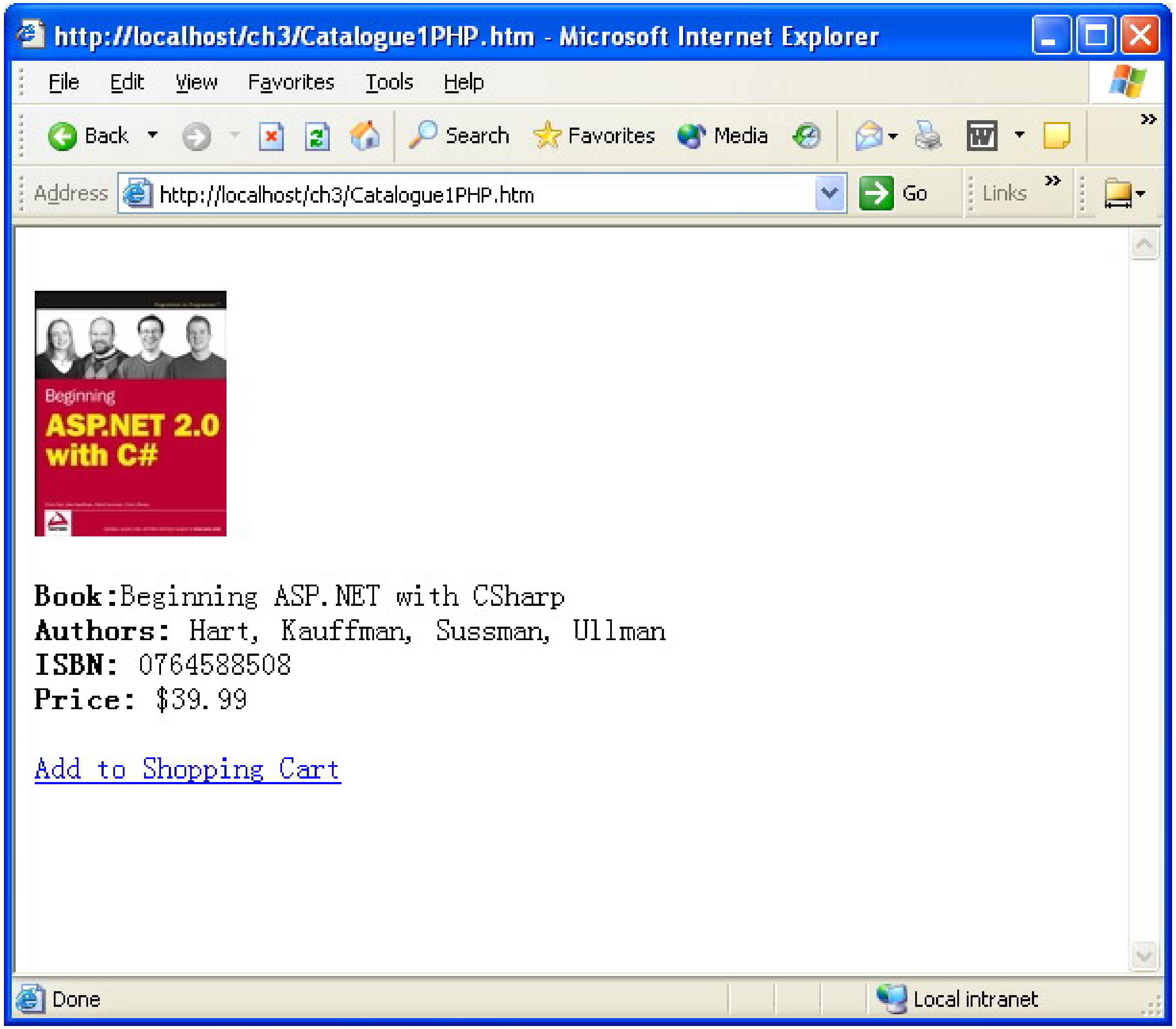

http://localhost/ch3/Catalogue1PHP.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch3/Catalogue1PHP.htm#
Go Links
Beginning ASP.NET 2.0 with C#
http://localhost/ch3/Catalogue1PHP.htm#
Local intranet

Book: Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: $39.99

Add to Shopping Cart

Beginning ASP.NET with CSharp 3 Remove Item

Clicking Remove Item


http://localhost/ch3/Catalogue1PHP.htm - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/ch3/Catalogue1PHP.htm
Go Links
Beginning ASP.NET 2.0 with C#
Done
Local intranet

Book: Beginning ASP.NET with CSharp
Authors: Hart, Kauffman, Sussman, Ullman
ISBN: 0764588508
Price: $39.99

Add to Shopping Cart

Maintaining the Shopping Cart

- The shopping cart has to be stored as persistent information between user interactions
- We can choose to store the details on the server or on the client
- If we keep the details on the client, then some information may need to be sent to the server on each invocation; client side storage is an option in this example
- We choose to use the server in this example. For the server, each time we interact is essentially a new call to the server – the PHP script will run over again, with a new invocation

Maintaining the Shopping Cart

- So – how do we keep persistent data on the server?
  - We could store it in an XML file or a database
  - Or – we can use a session variable which persists on the server throughout the **client** session that caused its creation
  - If we want persistence beyond the client session – eg if the user leaves the page in the browser, and it is desired that the information be restored with a subsequent client session – then for **server** storage, a file or database solution is necessary
- We choose to store the cart in a session variable in order to maintain the cart between successive accesses to the server.
  - Each session is unique to the user request that created it and can only be accessed by subsequent requests from the same user in the same user session (ie browser invocation).
  - So – we can't close down the client session, and later expect to pick up the shopping cart on the server!

Maintaining the Shopping Cart

- As mentioned, we choose to maintain the shopping cart on the server.
- For communication between server and client we represent the cart as an XML document

```
<?xml version="1.0"?>
<cart>
  <book>
    <title>Book Title</title>
    <quantity> 3 </quantity>
  </book>
</cart>
```

If there were more books in the catalogue, then there may be additional book items in the XML

- The cart has to be updated following each user-interaction
- We are adding just one book element, or removing it, or changing the quantity, but we should write code that can be readily modified to handle multiple books

Client Page – Catalogue1PHP.htm

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
    <script type="text/javascript" src="xhr.js"></script>
    <script type="text/javascript" src="CartPHP.js"></script>
</head>
<body>
    <br/><img id="cover" src="begaspnet.jpg" /><br /><br />
    <b>Book:</b><span id="book">Beginning ASP.NET with CSharp</span><br />
    <b>Authors: </b><span id="authors"> Hart, Kauffman, Sussman, Ullman</span>
    <br /><b>ISBN: </b><span id="ISBN">0764588508</span>
    <br /><b>Price: </b><span id="price">$39.99</span>
    <br /><br />
    <a href="#" onclick=”AddToCart()" >Add to Shopping Cart</a>
    <br /><br />
    <span id="cart" ></span>
</body>
</html>
```

Notes

- Note that we are storing the details of the book in the client (hard-coded)
- This is not suitable for a real application, where the book catalogue would be stored on the server, in a database or in XML form, with suitable identifying details retrieved by the client on loading, and then displayed in the client document, to enable the user to select books for purchase

Representing the Cart

- On the server : an associative array, that pairs the book title with the number of copies in the cart
- As data for transferring between server and client : as an XML document, to be sent from server as text, but picked up on the client (in the responseXML property of an XHR object) as an XML DOM object
- On the client : an XML DOM object, that can be read, and changed

CartPHP.js

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

function AddToCart()
{   var book  = document.getElementById("book").innerHTML;

    xhr.open("GET", "ManageCart.php?action=Add&book=" +
        encodeURIComponent(book) + "&value=" + Number(new Date), true);
    xhr.onreadystatechange = getData;
    xhr.send(null);
}

function DeleteFromCart()
{   var book  = document.getElementById("book").innerHTML;

    xhr.open("GET", "ManageCart.php?action=Remove&book=" +
        encodeURIComponent(book) + "&value=" + Number(new Date), true);
    xhr.onreadystatechange = getData;
    xhr.send(null);
}
```

Here the variable “book” stores the name of the single element with id “book” stored in the html document. If there are more books, this code will need to be altered

Number(new Date) is to force URL to be different from last time, so that IE aggressive cache doesn’t occur

The action “add” and the book name are passed to the server, so that the shopping cart can be altered there.

The PHP function in ManageCart.php will use the value of the action parameter to determine what processing to do

CartPHP.js

```
function getData()
{   if ((xhr.readyState == 4) &&(xhr.status == 200))
    {   var serverResponse = xhr.responseXML;
        var books = serverResponse.getElementsByTagName("book");
        var cartDisplay = document.getElementById("cart");
        cartDisplay.innerHTML = "";
        for (i=0; i<books.length; i++) // this will handle any number of books in the cart
        {  if (window.ActiveXObject) // IE uses "text"
           { cartDisplay.innerHTML += " " +books[i].firstChild.text;
             cartDisplay.innerHTML += " " + books[i].lastChild.text + " " + "<a href='#'
             onclick='DeleteFromCart'>Remove Item</a>";
           }
           else
           { cartDisplay.innerHTML += " " +books[i].firstChild.textContent; // WC3 uses textContent
             cartDisplay.innerHTML += " " + books[i].lastChild.textContent + " " + "<a
             href="#" onclick="DeleteFromCart()">Remove Item</a>";
           }
        }
    }
}
```

Here, by contrast, the variable "books" stores the array of book elements stored in the shopping cart, sent from the server

The firstChild of a book element gives the name of the book.

Maintaining the Shopping Cart

- As mentioned, we choose to maintain the shopping cart on the server.
- For communication between server and client we represent the cart as an XML document

```
<?xml version="1.0"?>
<cart>
  <book>
    <title>Book Title</title>
    <quantity> 3 </quantity>
  </book>
</cart>
```

If there were more books in the catalogue, then there may be additional book items in the XML

- The cart has to be updated following each user-interaction
- We are adding just one book element, or removing it, or changing the quantity, but we should write code that can be readily modified to handle multiple books

Notes

1. To access the text that represents the book name, the text stored in the first child node of the book has to be used. In IE (4+) this means using **text**, whereas in WC3-compliant browsers, textContent is used. (Note that IE8 does not have textContent. We need conditional code. (Eventually when everything is compliant, we can get rid of this nuisance!!))
2. When using the GET protocol, IE caches the page, and so the data passed will be the same as last time, unless the query string is changed. We force a change of query string by appending "&value=" + Number(new Date). Since the Date will be different on each call, the URL used in the GET will be different, and so the cached call will not be used.
3. An alternative is to use the POST protocol that would be a better approach, and you should write that alternative code.

ManageCart.php

$myCart is a PHP variable that is an associative array. It basically associates a quantity with each named Book title.

```
<?php
    //session_register("Cart"); // registers a session variable called Cart
    session_start(); //replace the above with this line
    header('Content-Type: text/xml');
?>
<?php
    if(!isset($_SESSION["Cart"])) $_SESSION["Cart"] = ""; // add this line
    $newitem = $_GET["book"]; // book name
    $action = $_GET["action"];  // add or remove?
    if ($_SESSION["Cart"] != "") // the “cart” already exists with an item in it
    {
        $myCart = $_SESSION["Cart"]; // assign the session variable to $myCart
        if ($action == "Add") // we are processing an “Add” request
        {
            if ($myCart[$newitem] != "") // the cart already has this book in it
            {
                $myCart[$newitem]++; // add 1 to no of copies
             }
            else // this is the first copy of this book to be added (there may be other books)
```

ManageCart.php (Cont'd)

```
        else // this is the first copy of this book to be added (there may be other books)
         {
            $myCart[$newitem] = “1";
          }
    }
    else // we are processing a “Remove” request
    {
        $myCart= ""; // nb – we are clearing the whole cart; this is what the spec requires
    }
}
else // the “cart” is NOT present – ie, we have no books ordered; order one!
{
    $myCart[$newitem] = "1";
}
// copy modified cart to session variable, convert to serialized XML and send to client
$_SESSION["Cart"] = $myCart;
echo (toXml($myCart)); // call function toXML
```

Notes

1. We first register this session, and give it a name “Cart”
2. We also set the type of information to be returned (text/xml)
3. We then pick up the information passed from the client – the book details and whether we are adding or removing an item
4. And then deal with the following cases :
    1. The cart is currently not empty
        1. We are adding an item
            1. The cart already has THIS book in it
            2. The cart does not already have THIS book in it
        2. We are removing an item
            - Note that here we just clear the whole cart (if spec requires that we just cut one copy, different code is needed)
    2. The cart is currently empty

Notes

5. We deal partly with the possibility that the book catalogue has more than one book; we correctly add a book (whether another copy of a book in the cart, or a new book not yet in the cart), but when we delete an item, the code currently deletes the whole cart. This is not right if the cart contains more than one book.
6. We conclude by saving the changed cart back to the session variable (to be picked up on next user-interaction), and then convert the cart to XML, serialize this as a string, and send to the XHR object

ManageCart.php (Cont’d)

```
    function toXml($aCart)
    {   $doc = new DomDocument('1.0');
        $cart = $doc->createElement('cart');
        $doc->appendChild($cart);
        foreach ($aCart as $titleValue => $qtyValue)
        {    $book = $doc->createElement('book');
             $cart->appendChild($book);
             $title = $doc->createElement('title');
             $book->appendChild($title);
             $value = $doc->createTextNode($titleValue);
             $title->appendChild($value);
             $quantity = $doc->createElement('quantity');
             $book->appendChild($quantity);
             $value2 = $doc->createTextNode($qtyValue);
             $quantity->appendChild($value2);
        }
        $strXml = $doc->saveXML(); // this serializes the XML as a string
        return $strXml;
    }
?>
```

This is how to loop through the items of an associative array

```
<?xml version="1.0"?>
<cart>
  <book>
    <title>Book Title</title>
    <quantity> 3 </quantity>
  </book>
</cart>
```

foreach (array_expression as $key => $value)

foreach ($aCart as $titleValue => $qtyValue)

$aCart["Beginning ASP.NET with Csharp"] = 3

$titleValue ← "Beginning ASP.NET with Csharp"

$qtyValue ← "3"

Thank you!