



Web Application Development : An Overview and Introduction to Web Application Development

Week 1



SWINBURNE
UNIVERSITY OF
TECHNOLOGY

Content of this Lecture

- Unit overview
- Web applications, architectural patterns and models
- Classic web application model
- A glance at a PHP example
- The Ajax application model
- Ajax component technologies
- A glance at an Ajax example

Staff

■ Convenor & Lecturer, Tutor

□ Dr Wei Lai, wlai@swin.edu.au

□ Dr Md Kafil Uddin, mdkafiluddin@swin.edu.au

□ Dr Rao Liaqat, rliaqat@swin.edu.au

□ Dr Muhammad Islam, muhammadislam@swin.edu.au

□ Baqer Mir Mohammed, bmirmohammed@swin.edu.au

Pre-requisites

- COS60004 Creating Web Applications

PLUS

COS60006 Introduction to Programming or
COS60010 Technology Inquiry Project

OR

- Admission into the Master of Information Technology (professional Computing) course.

Teaching Materials

- Lecture Slides are available on Canvas.
- I recommend that you download and print and bring to lectures.
- I will use Canvas for all communication about the unit.

Assessment – see Unit Outline

- Weekly lab exercise and check
 - 2% for each lab exercises, **checked at the end of the lab or the beginning of the next lab;**
 - 11 labs (the best 10 labs selected & capped at 20%)

- Project 1 (35%)

- Project 2 (45%)

- To pass this unit, you must achieve:
 - *an aggregate mark for the subject of 50% or more.*

Extensions

- Extensions must be applied for before the due date by emailing the unit convener, and **a medical certificate must be provided at the time of request.**
- A request without a medical certificate will not be responded to.
- Extensions are generally granted up to a maximum of **7 days.**

Extensions

swinburne.edu.au/life-at-swinburne/student-support-services/special-consideration-assist...

Extensions

If your preparation for a non-exam assessment is affected by illness or other extraordinary circumstances outside your control, you may apply in advance for an extension to the due date.

Extensions are generally granted up to a maximum of seven days. In very special circumstances a longer extension may be given.

Late Submission

- Plan your time and submit early. Do not leave it to the last minute.
- Back up your work frequently
- Computer failure, house moving, personal travel, holidays are not excuses for extension.
- Extensions must be applied for before the due date by emailing the unit convener, and a medical certificate must be provided at the time of request.
- Late Submissions - Unless an extension has been approved, late submissions will result in a penalty. You will be penalised 10% of the assessment's worth for each calendar day the task is late, up to a maximum of 5 days. After 5 calendar days, a zero result will be recorded.

Group work guidelines

- In this unit, students are encouraged to work on lab work with ONE other student.
- However;
 - ☐ Each student must independently write up and submit their work.
 - ☐ Where a student cannot explain work submitted, it will be awarded 0 marks.

Recommended Reading Materials

- PHP/MySQL - Gosselin, Kokoska and Easterbrooks, *PHP Programming with MySQL, 2nd Edition*, Course Technology, 2011.
- Ajax – Ullman and Dykes, *Beginning Ajax*, WROX, Wiley Publishing, Inc., 2007.
- On-line books / eBooks / hard copies:
 - ☐ There are many good books on-line, as eBooks, available through the Library. <http://www.swinburne.edu.au/lib/>
 - ☐ Hard copies of the above book are also reserved in the library.
- On-line References:
 - ☐ Primary references about the Web, Web Servers, PHP, and MySQL are online: see “Blackboard” resources for some useful links.

Labs

- Each student will have an allocated lab
- Labs start in Week 1 with in-lab exercise completion check
- Lab marks count (20%) !

Web Applications

- This unit is all about how to BUILD **web applications**
- We will focus on the technologies & techniques used
- We will NOT focus very much on “good interface design”, although in projects I will expect students to produce systems with “decent” interfaces! (I assume that most, if not all, students, have studied or will study the unit on “Usability”.)

Practical Work

- This is a DEVELOPMENT unit
- You will only achieve the desired outcomes if you write lots of programs!
- Documentation is also important to help you and others understand the structure and behaviour of your programs!
- You really must do all the practical work, and you really must do the projects yourself!
- The projects are designed to help you learn
- **Warning:** We will carefully check for unauthorised collaboration on projects, with students enrolled in this semester or previous semesters, and will apply the most serious penalties allowed by the regulations

Technology

- Use mercury (Web server) for project work; I have arranged accounts for all enrolled students
- You can set up Apache, PHP & MySQL on your own PC or Mac – I will not give you instructions on how to do this, but strongly suggest that you do it in order to get easy access to a test-bed for your practical work. Consider using XAMPP – probably the easiest way to do this. (Just google “XAMPP”)
- Use the discussion group on Canvas to share knowledge about this with one another – that’s your best bet if you are having difficulties in getting things to work!
- You can use whatever tools you like – get a good web editor

Provisional Schedule: refer to unit outline

Teaching Style

■ Interactive:

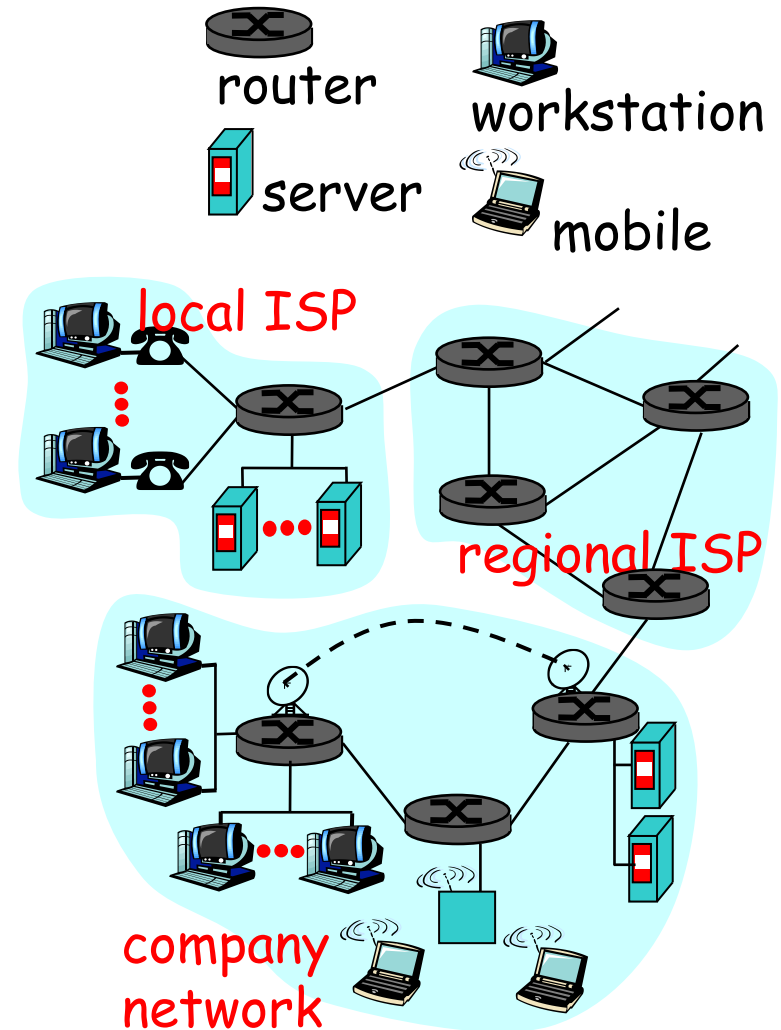
- ☐ I will ask you questions at certain points to help you think deep; these questions may appear in the final exam; so be active and attentive!
- ☐ You are welcome to ask questions in class, if time allows, I will try to give you a detailed answer, otherwise let's discuss offline

■ Examples and demonstrations

- *Don't forget to turn off/down your mobile phone or other computing equipment*

Review : The Internet

- ❑ millions of connected computing devices:
hosts = end systems
- ❑ running *network apps*
- ❑ *communication links*
 - ❑ fiber, copper, radio, satellite
 - ❑ transmission rate = *bandwidth*
- ❑ *routers*: forward packets (chunks of data)
- ❑ *protocols* control sending, receiving of msgs
 - ❑ e.g., TCP, IP, HTTP, FTP, PPP
- ❑ *Internet*: “network of networks”
 - ❑ loosely hierarchical
 - ❑ public Internet versus private intranet



Review : Web and HTTP

- **Web page** consists of **objects**
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of **base HTML-file** which includes several referenced objects
- Each object is addressable by a **URL**
- **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

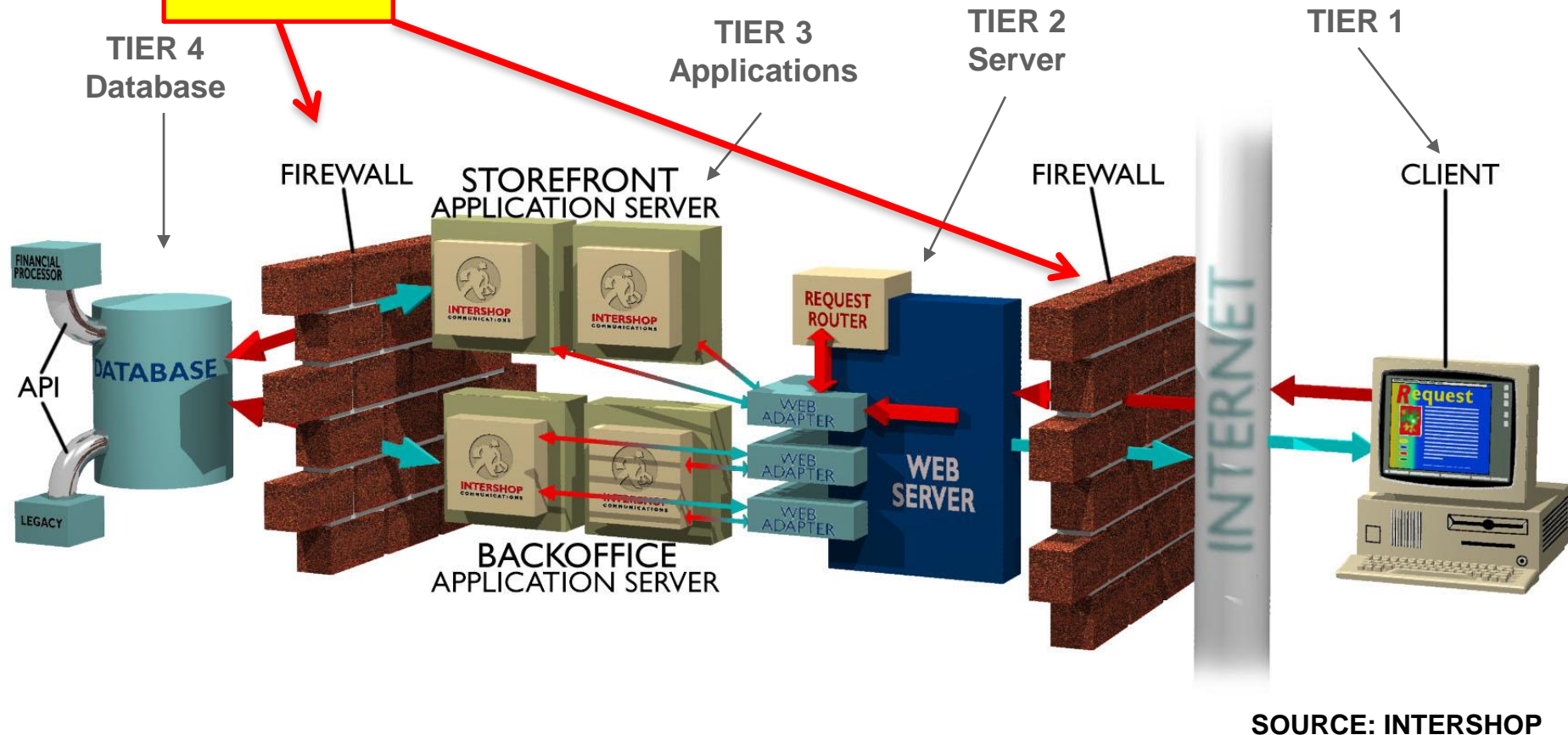
protocol://**domain_name**/**path_name**

http://**www.swin.edu.au**/**ict/research/citr/wde/index.php**

Review: Web Architecture

How are web sites constructed?

NB : We will not focus on security in this unit.



A Simple View of Web Apps

- User clicks on a URL in a browser, which points to a file on a web server
- Client-side page is downloaded from server, and is loaded within web browser
- User interacts with this page
- Interaction causes requests to be sent to server, to execute programs on the server – or possibly even sends requests to external sources (eg Google, to access Google maps)
- The server processes these, and sends responses, (including data in various formats), back to the client
 - This may include the server going off to other external places, eg a database server, or some third-party provider
 - It may potentially involve the server-side program itself being distributed to run on a grid or cloud computing installation, invisible to the user
- Client receives this data, processes it, does whatever it needs to, and then gets ready for next interaction. This *may* result in the client page being modified.

A Simple View of Web Apps

- We want all of this to be smooth
 - No jerky visuals
 - No user hold-ups waiting for server response
 - User can continue to interact with the user-interface whilst waiting for response (as encountered in many popular web systems – eg Google)
- We want all the user-interaction modes that we are used to on PC software to be available
 - Mouseovers, animations, dynamic menus, command completions, etc

The Challenges of Web Development

- How do we partition processing between client and server?
 - What should we do on the server?
 - What should we do on the client?
- What judgements should we make about load, security, technologies available on each side?
- How should we manage the server-side data so that it is sent to the client in an appropriate form?
- How should we manage the data when it reaches the client?
- What languages and technologies should we use?
- How do we deal with different web browsers on the client side?

Technologies

- The network – indeed the whole internet - is the computer!
- We will use the following major technologies
 - Client-side (universal)
 - Browser (which provides a virtual machine)
 - (X)HTML and JavaScript
 - XML
 - Server-side (example)
 - Principally PHP, XML and Database (e.g., MySQL)
 - Alternatives to PHP are ASP.NET, Java

Architectural Pattern for Application Design

- Model View Controller (MVC) Model
 - **Model** is the **domain**-specific representation of the information on which the application operates. Many applications use a persistent storage mechanism (such as a *database*) to store data
 - **View** renders the model into a form suitable for interaction, typically a *user interface* element
 - **Controller** processes and responds to events, typically user actions, and may invoke changes on the model
- MVC separates **model** and **view** by decoupling *data access* and *business logic* from *data presentation* and *user interaction*, by introducing an intermediate component: the **controller**

MVC and Web Applications

- Many web application frameworks follow the MVC to separate the *data model*, *business rules* and *user interface*.
- In a web application
 - the **view** is the actual *HTML* page (for user input)
 - the **controller** is the code which gathers dynamic data and generates the content within the HTML (handles the input event, accesses/updates the model)
 - the **model** is represented by the actual content, usually stored in a *database* or *XML* files, with various operations available.

Fat Client vs. Thin Client

- Web systems follow the model of Client/Server computing – basically the browser handles the user interface and a server (or servers) handles the model (data and logic). However, with web applications, the division of labour is not quite so simple.
- For business logic, there are two basic options:
 - **Fat** Client: the browser does most of the computational work, using scripting languages (JavaScript). The server mainly hosts the data, and serves it up to the client for processing.
 - **Thin** client: the server at the other end does most of the processing, as well as hosting the data, using server-side technologies such as ASP.NET, JSP and PHP.

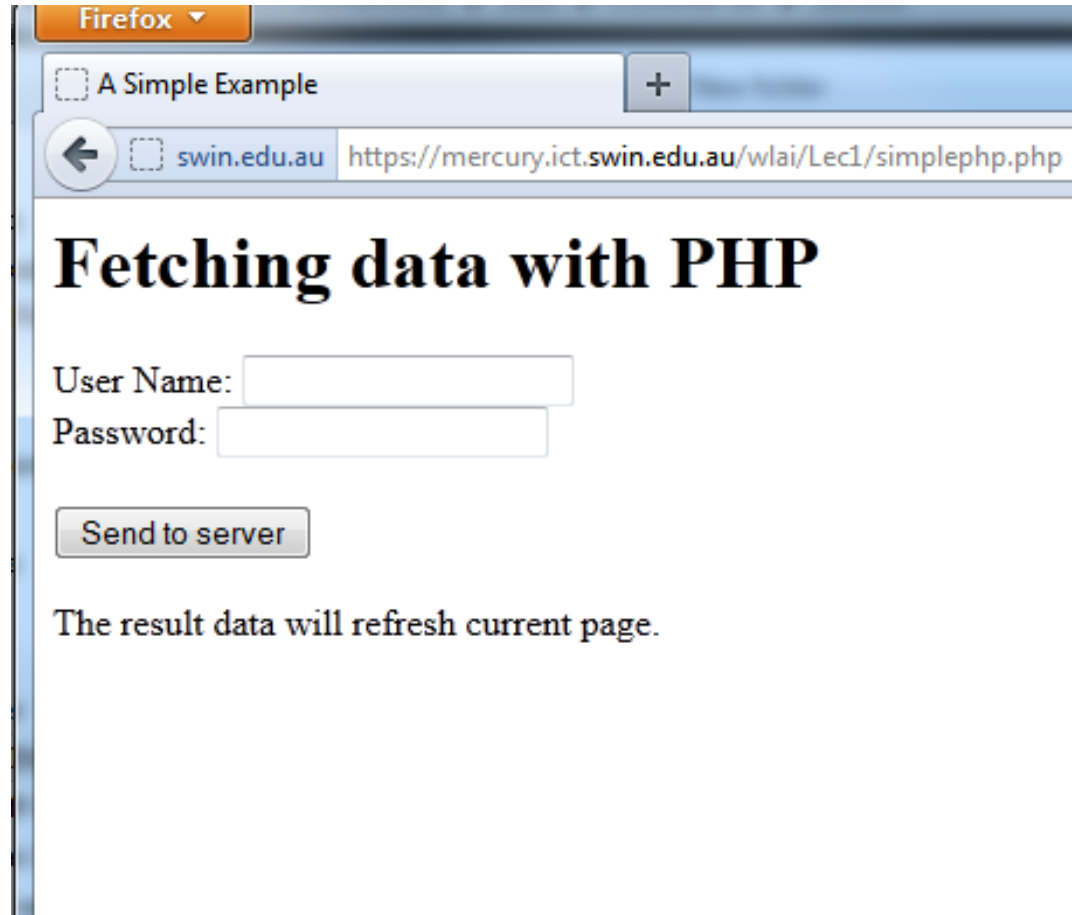
Fat Client vs. Thin Client

- Determining where to host the processing of data is a decision-problem for all systems
- A number of issues need to be considered
 - ❑ How busy is the server? Does it have the capacity to host the processing from many concurrent users?
 - ❑ How much data needs to be transmitted if processing is on the client?
 - ❑ Is security an issue, making minimised transmission of data recommendable?
 - ❑ How complex is the program logic – in particular, to what extent is it made complex by the need to consider possible browser differences?

Classic Web Application Models

- are broadly based on the server-side method
 - ☐ enter your inputs
 - ☐ send the page to the server, which processes the inputs, and then sends a new page back to be rendered on the client
 - ☐ The user waits for a response before the next client action
- This is a “click, wait, and refresh” user interaction model
- It is a synchronous “request/response” communication model; the user waits idly for the server to respond
- There are some problems
 - ☐ slow performance: full page replacement; waiting synchronously
 - ☐ Limited interactivity

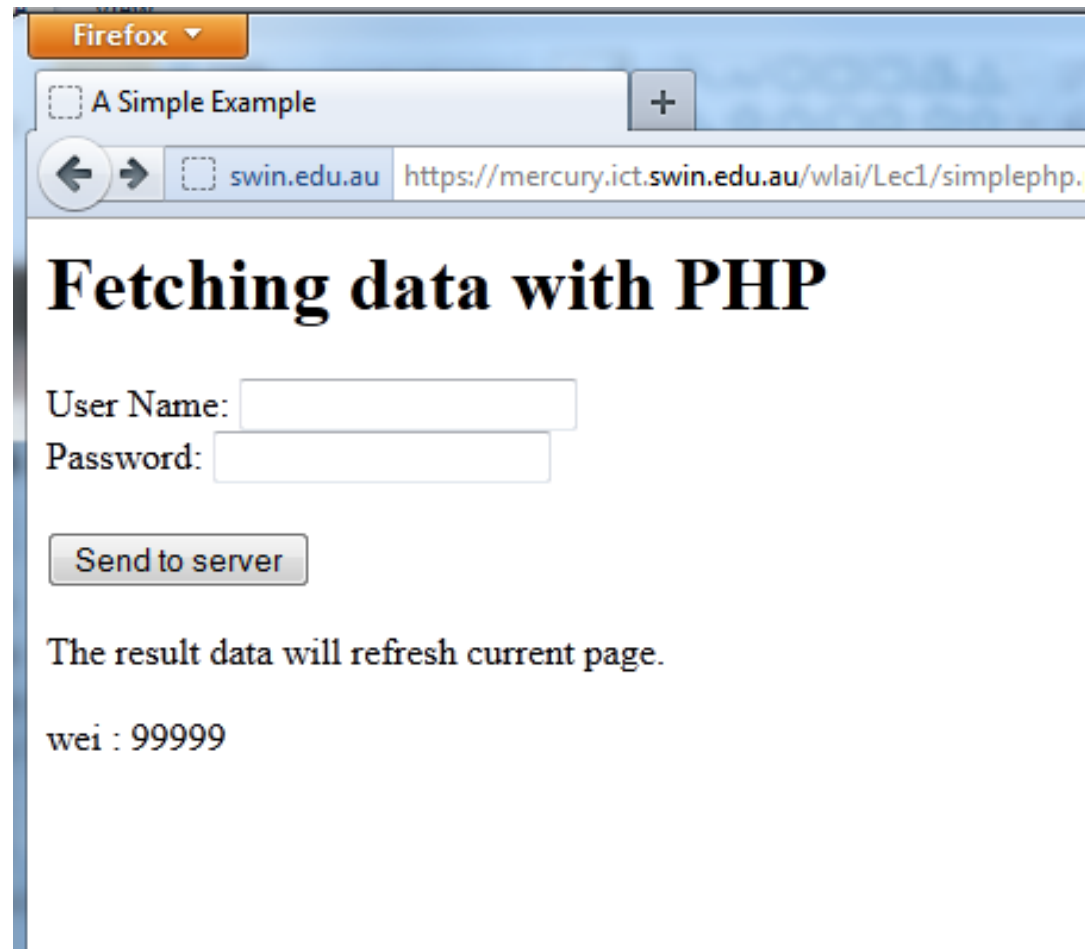
A Simple Example – Fetching Data with PHP



The screenshot shows a Firefox browser window with a single tab titled "A Simple Example". The address bar displays the URL `https://mercury.ict.swin.edu.au/wlai/Lec1/simplephp.php`. The page content features a heading "Fetching data with PHP" in a large, bold, black serif font. Below the heading are two input fields: "User Name:" followed by a text box, and "Password:" followed by a text box. A "Send to server" button is positioned below the password field. At the bottom of the form area, a text label reads "The result data will refresh current page."

<https://mercury.swin.edu.au/wlai/Lec1/simplephp.php>

Show the Result in a New Page



The screenshot shows a Firefox browser window with a single tab titled "A Simple Example". The address bar displays the URL "https://mercury.ict.swin.edu.au/wlai/Lec1/simplephp.". The page content includes a heading "Fetching data with PHP", followed by two input fields labeled "User Name:" and "Password:". Below these fields is a button labeled "Send to server". Underneath the button, there is a text line "The result data will refresh current page." and another line "wei : 99999".

Firefox ▾

A Simple Example +

← → swin.edu.au https://mercury.ict.swin.edu.au/wlai/Lec1/simplephp.

Fetching data with PHP

User Name:

Password:

The result data will refresh current page.

wei : 99999

User Interface

```
<!-- file simplephp.php -->
<HTML XMLNs="http://www.w3.org/1999/xhtml">
  <head>
    <title>A Simple Example</title>
  </head>
  <body>
    <H1>Fetching data with PHP &nbsp;  </H1>
    <form method="get">
      <label>User Name: <input type="text" name="namefield"> </label>
      <label><br>Password: <input type="text" name="pwdfield"> <br><br></label>
      <input name="submit" value = "Send to server">
    </form>
    <p> The result data will refresh current page.</p>
  </body>
```


Embedded PHP

```
<?php
    // get name and password passed from client
    if(isset($_GET['namefield']) && isset($_GET['pwdfield']))
    {
        $name = $_GET['namefield'];
        $pwd = $_GET['pwdfield'];
        // sleep for 5 seconds to slow server response down
        sleep(5);
        // write back the password concatenated to end of the name
        ECHO ($name." : ".$pwd);
    }
?>
</HTML>
```

In PHP, a dot represents string concatenation. So this concatenates the name, with a colon, with the password

Discussion

- It's simple! PHP code embedded in HTML (simplephp.php)
- User enters data in a simple form
- User clicks on button
- The client sends the data as “parameter” to call the embedded PHP, and waits there
- The PHP “executes” on the server extract the data sent, concatenates the two data fields (we could do more here!), and sends a **new page** back! You cannot see the context information (i.e., the inputted user name and password)

Not Embedded PHP

```
<!-- file simpleForm.htm -->
```

```
<html>
```

```
<body>
```

```
<form action="getInfo.php" method="get">
```

```
User Name: <input type="text" name="namefield "><br>
```

```
Password: <input type="text" name=" pwdfield "><br>
```

```
<input type="submit">
```

```
</form>
```

```
</body>
```

```
</html>
```

getInfo.php

```
<html>
<?php
    // get name and password passed from client
    if(isset($_GET['namefield']) && isset($_GET['pwdfield']))
    {
        $name = $_GET['namefield'];
        $pwd = $_GET['pwdfield'];
        // sleep for 5 seconds to slow server response down
        sleep(5);
        // write back the password concatenated to end of the name
        ECHO ($name." : ".$pwd);
    }
?>
</html>
```

In PHP, a dot represents string concatenation. So this concatenates the name, with a colon, with the password

Embedded PHP

To show the values in the input fields after the user hits the submit button

Name: <input type="text" name="name" value="<?php echo \$name;?>">

E-mail: <input type="text" name="email" value="<?php echo \$email;?>">

Website: <input type="text" name="website" value="<?php echo \$website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo \$comment;?></textarea>

Ajax

- Various techniques have been devised to resolve some of the issues of the classic web model
- The aim is to allow dynamic change in the interface presented to the user, and to allow the user to continue interacting with the system whilst the server is busy responding to an earlier request
- “AJAX” – **A**synchronous **J**avaScript **a**nd **X**ML – now refers to a collection of technologies that allow the above.

The Ajax Application Model

- Original web page remains displayed in the browser
- Messages are sent to server to do some processing, asynchronously (ie the browser does not wait for response, and can still be used for other tasks before data comes back from server – but beware : the user may do something that will interfere with the previous request!)
- Results are sent back from the server when they are ready (as text or XML)
- JavaScript in the browser is essentially waiting for these, and when the data arrives back, it interprets these, and updates sections of the web page that is being displayed – triggered by receipt of the data from the server

Ajax : How

- There are several options available to implement the Ajax Model
 - XMLHttpRequest Object – our main focus, as this is now the standard technology used to manage asynchronous communication with the server, and is designed to handle XML
 - Other options: Hidden Frames, Hidden iFrames (not introduced in lecture)

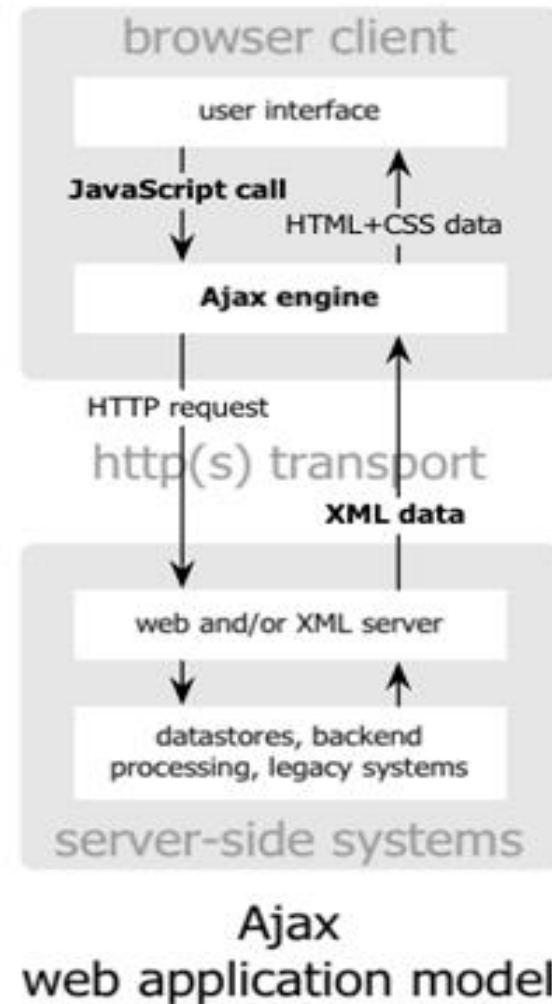
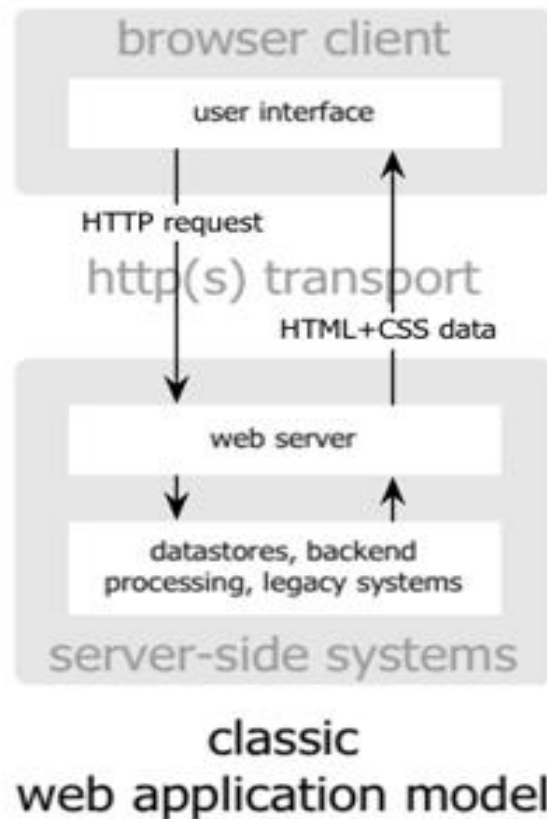
The Ajax Application Model – User Interaction

- “Partial screen update” user interaction model
 - During user interaction within an AJAX-based application, only user interface elements that contain new information are updated; the rest of the user interface remains displayed without interruption.
 - This "partial screen update" interaction model not only enables the continuous operation context, but also makes non-linear workflow possible.
 - The partial update is effected through dynamic change of the internal representation of the “document” displayed in the browser

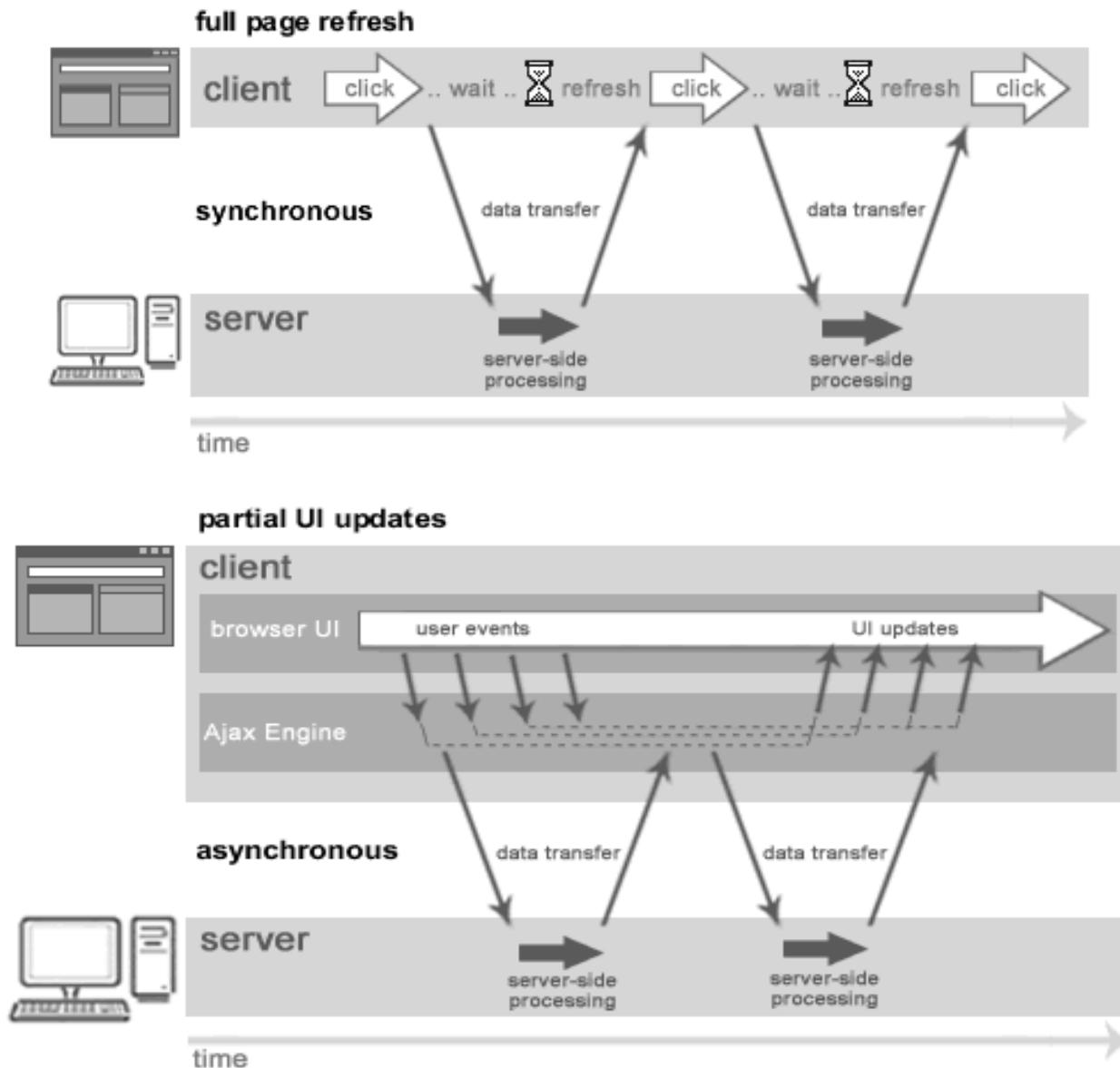
The Ajax Application Model - Communication

- Asynchronous communication model
 - For an AJAX-based application, the request/response can be asynchronous, decoupling user interaction from server interaction. As a result, the user can continue to use the application while the client program requests information from the server in the background.

Ajax vs. Classic Web Application Models



Ajax vs. Classic Communication Models



Popular Examples using Ajax

- Yahoo's flickr

<http://www.flickr.com/>

- Basecamp

<http://www.basecamphq.com/>

- Amazon

<http://www.a9.com/>

- Google Suggest

<http://www.google.com/webhp?complete=1&hl=en>

- Google Maps

<http://maps.google.co.uk/>

Ajax Component Technologies

- **XHTML** and **CSS** for *standards-based presentation*
- **Document Object Model** for *dynamic display and interaction on the client, and management of XML on both client and server*
- **XML** and **XPATH/XSLT** for *data interchange and manipulation*
- **XMLHttpRequest** for *asynchronous data retrieval*
- **JavaScript** for *binding everything together*
- and also need a server-side language to handle any interaction with the server (**PHP**, ASP.NET*, or Java)

XHTML

- (XHTML) - the more exacting version of HTML.
- HTML standard specified as an XML document.
- HTML is fairly easygoing, XHTML is more strict and follows XML's rules
 - well formed (tags are correctly opened and closed, and nested)
 - is required to use the DOM (Document Object Model)

CSS

- Cascading Style Sheets - describe the presentation and layout of the text and data contained within an HTML page.
- web application design criteria - clear division between the content/structure of the page and the presentation.
- changes made to the style sheet are instantly reflected in the display of the page.
- linked into the document commonly with the HTML <link> tag,
- possible to specify style attributes for each individual HTML tag on a page.
- can also access CSS properties via the DOM.

DOM

- The Document Object Model – API
- lets developers create and modify HTML/XML documents as sets of program objects (in a hierarchy), which makes it easier to design web pages that users can manipulate.
- defines the attributes associated with each object, as well as the ways in which users can interact with objects.
- works with JavaScript, XHTML, and CSS to dynamically change the appearance of Web pages, and make Ajax applications particularly responsive for users.
- DOM techniques are applicable to the client side, and as a result, the browser can update the page or sections of it instantly.
- Supported by most browsers: IE, Firefox, Safari, Chrome and Opera

JavaScript

- interacts with HTML code and makes web pages and Ajax applications more active
- can be embedded in HTML pages
- Ajax uses asynchronous JavaScript, allows an HTML page calls the server asynchronously, retrieve new XML data, and simultaneously update the web page partially in the background, all while the user continues interacting with the program.
- JavaScript is a cross-platform scripting language → Ajax applications require no plug-ins

XML

- the standard markup language that is used to describe and structure data exchanged on the internet
- allows developers to *customise* own elements (i.e., tags) to structure data and provide it meaning, while HTML has pre-defined tags and fixed tag semantics
- an XML document contains no information about how it should be displayed or searched; it needs other technologies to search and display information from it
- Ajax uses XML to encode data for transfer between a server and a browser (although plain text can also be used)

XML Technologies

- XML documents – have to be well-formed
- DTD and XML Schema – define what is a “valid” document
- CSS
- DOM
- Xpath - a language for selecting parts of an XML document according to criteria
- XSLT - a language for transforming XML documents into another format (e.g. XHTML, XML, or text)
- XQuery
-

XMLHttpRequest (XHR) Object

- plays a major role in Ajax applications
- allows a developer to transport the data backward and forward behind the scenes
- Can provide the asynchronous part of the Ajax application by using the *onreadystatechange* event to indicate when it has finished loading information.
- IE 5/6 uses ActiveX control called the XMLHttpRequest object
`var xhr = new ActiveXObject("Microsoft.XMLHTTP");`
- Firefox, Netscape 7, Safari 1.2+, Opera, Chrome and IE 7+ use a "native" XMLHttpRequest object
`var xhr = new XMLHttpRequest();`

Server-Side Technologies

- PHP
- ASP.NET
- J2EE
- We will learn PHP programming in week 2 and week 3, and manipulating MySQL databases with PHP in week 4

Other Technologies Examined

- Web services and APIs
 - Reuse of existing software components (written by others)
- Patterns
 - A solution to a common problem
 - Reuse again!
- JSON – JavaScript Object Notation
 - Alternative to XML; send JavaScript objects in text form, and evaluate in browser
- Ajax Frameworks

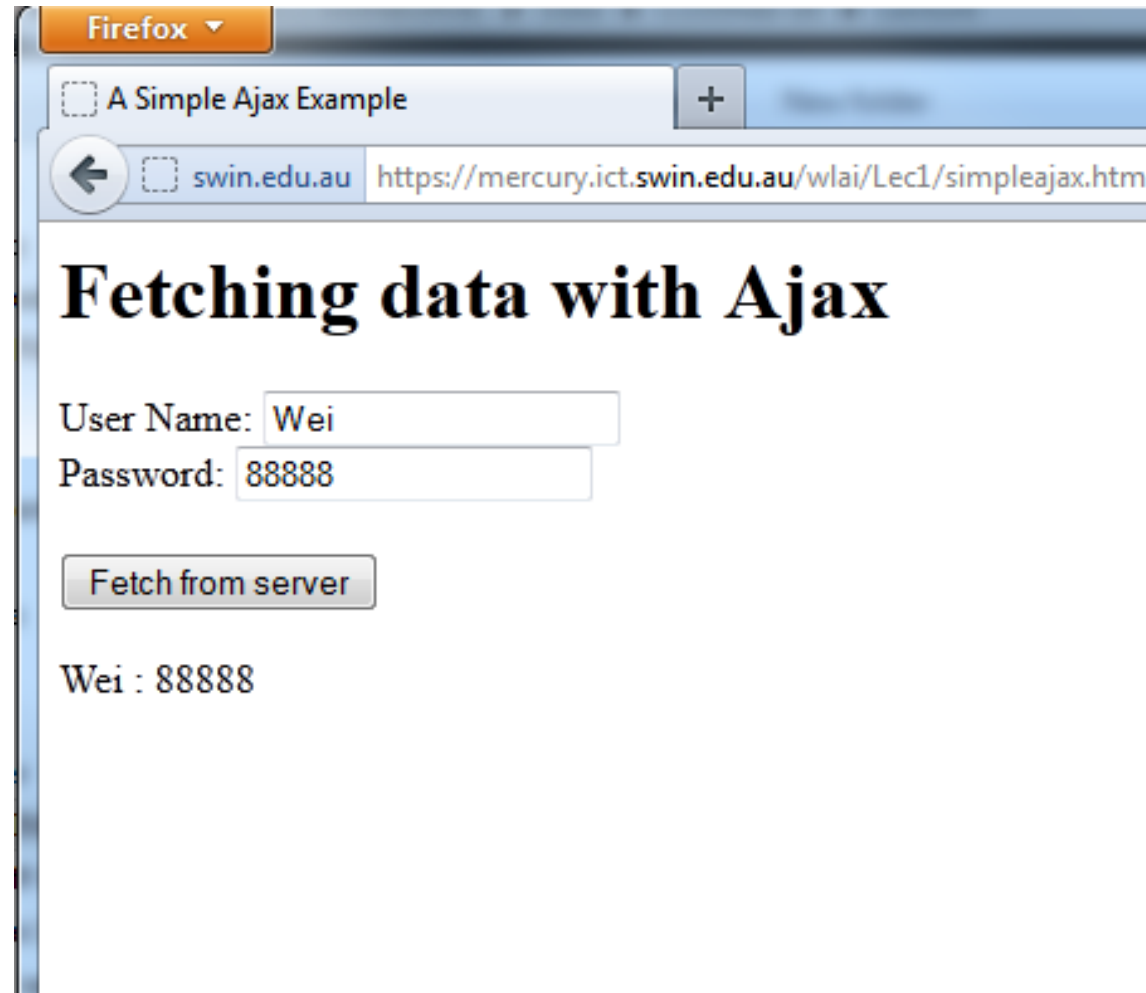
Reconsider the Simple Example in Ajax



The screenshot shows a Firefox browser window with a single tab titled "A Simple Ajax Example". The address bar displays the URL "https://mercury.ict.swin.edu.au/wlai/Lec1/simpleajax.htm". The page content includes a heading "Fetching data with Ajax", followed by two input fields labeled "User Name:" and "Password:". Below these fields is a button labeled "Fetch from server". At the bottom of the form area, there is a text label "The fetched data will go here.".

<https://mercury.swin.edu.au/wlai/Lec1/simpleajax.htm>

Show the Result on the Same Page



Behind the Scenes, as the User Imagines It

- User goes to URL (<http://mercury.swin.edu.au/wlai/Lec1/simpleajax.htm>)
- User enters data in a simple form
- User clicks on button
- Onclick event handler for button causes data to be sent to server as “parameter” to call of the URL of the PHP file (<http://mercury.swin.edu.au/wlai/Lec1/data.php>)
- This PHP file “executes” on the server extract the data sent, concatenating the two data fields (we could do more here!), and sends it back to the client
- When client has received the data, it displays the data received from the server in a “div” set aside in the browser document for that purpose – thus the document is itself changed

What Do We Have to Develop?

- Main (X)HTML file to display user interface, to handle interaction and to display outcome
- XHR object creation function (in an external JavaScript “library” function to handle browser differences in creating an XHR object) - the XHR object is used for handling communication, i.e., pass some data to the server, and later receive back some data created by a PHP program on the server.
- JavaScript file to drive the client side – react to user input, communicate with server, process data returned from server
- PHP script on server to receive data from client, process it, and send data back to client
- Note that all files reside in same directory on the server – we will see later that *data* stored on the server should be in a separate directory, for access control purposes.

“Main” HTML File

```
<!-- file simpleajax.htm -->
```

```
<HTML XMLNs="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<title>A Simple Ajax Example</title>
```

```
<script type="text/javascript" src="xhr.js"></script>
```

```
<script type="text/javascript" src="simpleajax.js"> </script>
```

```
</head>
```

```
<body>
```

contains XHR
object creation
function that
caters for browser
differences

To follow on
next slide

JavaScript code
for this example

“Main” HTML File

```
<body>
  <H1>Fetching data with Ajax &nbsp;  </H1>
  <form>
    <label>User Name: <input type="text" name="namefield"> </label>
    <label><br>Password: <input type="text" name="pwdfield"> <br><br> </label>
    <input name="submit" type = "button" onClick = "getData('data.php',
      'targetDiv', namefield.value, pwdfield.value) " value = "Fetch from server">
  </form>
  <div id="targetDiv">
    <p>The fetched data will go here.</p>
  </div>
</body>
</HTML>
```

getData is in the file simpleajax.js

When button is clicked, execute function **getData**, telling it : which PHP file to execute, where to place the data that comes back from the server , and what data to send to PHP

Discussion : Normal Synchronous Processing

- Suppose that the system used conventional synchronous processing.
- When the button is pressed in the user-interface, the event-handler attached to the button simply has to make a direct *remote procedure call* to the server, and then process the received data appropriately. The parameters passed would be the name and the password.
- When the remote procedure is called, the client would then WAIT for the remote procedure to return the result of its processing, as always happens in normal sequential (synchronous) computing. The fact that the called procedure is remote, on another machine, is irrelevant.
- The remote procedure on the server would accept the parameters sent to it, would concatenate the strings, and would RETURN the resulting string as the result of the procedure call.
- When the procedure result was received back at the client, as a result of the procedure call terminating, it would be assigned to the appropriate place in the document (within the code of the button event-handler).

Discussion : Asynchronous Processing

- With the asynchronous model, it works in a different way!
- This time, when the button is pressed, the event-handler should make the remote procedure call to the procedure on the server, but the event-handler should then immediately terminate, so that the client may accept on-going interaction from the user whilst the server is doing its work.
- We need to specify what is to occur once the remote procedure has sent its result back, as the client is not simply waiting for that to happen. The client will accept further user interaction, BUT IS ALSO WAITING TO RESPOND TO SIGNALS THAT THE SERVER PROCESSING HAS BEEN COMPLETED.
- The way this is done is that the event-handler must not only make the call to the remote procedure on the server, but also it must specify how the client is to respond when the result from the server is eventually received by the client.
- In other words, we need to specify an event-handler for the client-side event that it has received a message that the server has signalled it has done its work.

Discussion : Asynchronous Processing

- How does the client handle the returned data?
- The way we manage this is that we use an XMLHttpRequest object (XHR object) that sits in the client and serves as the communication link between the client and the server.
- We create this object, and set up the link to the file on the server that has to be “called”, and include the parameters that have to be passed to it – it’s a bit like plugging a device into a power outlet.
- We then specify what is to occur when the result from the server is eventually received by the XHR object. This is specified as an event-handler attached to the onreadystatechange event of the XHR object
- Finally, we initiate the remote call – this is a bit like switching the device on.

Discussion : Asynchronous Processing

- Note that in the client, the browser is forever monitoring the objects that have associated events. Thus for example, button presses are picked up.
- The XHR object is also monitored. Thus the **onreadystatechange** event of this object is monitored, and if this event occurs (or “fires”), the event-handler for THIS event executes.
- Unfortunately this event simply signifies that the **readyState** property of the XHR object has changed. It does not directly signify that the data has been received from the server.
- The event-handler thus must check if the data HAS been received, and that processing has completed properly. If not, it should do nothing. If so, it should pick up the received data, and process it as required.
- IT'S TIME FOR CODE!!

Our JavaScript Code : simpleajax.js

```
var xhr = createRequest(); // from file xhr.js

function getData(dataSource, divID, aName, aPwd) {
    if(xhr) {
        var place = document.getElementById(divID);
        var url = dataSource+"?name="+aName+"&pwd="+aPwd;
        xhr.open("GET", url, true);
        xhr.onreadystatechange = function() {
            alert(xhr.readyState); // to monitor progress
            if (xhr.readyState == 4 && xhr.status == 200) {
                place.innerHTML = xhr.responseText;
            } // end if
        } // end anonymous call-back function
        xhr.send(null);
    } // end if
} // end function getData()
```

Only do this if xhr was successfully created. If it was not, then in fact nothing happens! There is no connection to the server, no data returned, and the user will see nothing! To be honest, we should include an **else** alternative, that alerts the user to an error.

Use GET protocol. The "true" means that asynchronous access is requested.

Alternative, with named call-back function

```
var xhr = createRequest(); // from file xhr.js

function getData(dataSource, divID, aName, aPwd) {
    if(xhr) {
        var place = document.getElementById(divID);
        var url = dataSource+"?name="+aName+"&pwd="+aPwd;
        xhr.open("GET", url, true);
        xhr.onreadystatechange = placeData(place);
        xhr.send(null);
    } // end if
} // end function getData()
```

Here we provide a named function as the call-back, and supply a parameter as we do need to pass the location for the data display in to this function

```
function placeData(location) {
    alert(xhr.readyState); // to monitor progress
    if (xhr.readyState == 4 && xhr.status == 200) {
        location.innerHTML = xhr.responseText;
    } // end if
}
```

We only ever use this function for the one purpose, and it really does not need to be named. So the typical programming practice is to use the anonymous function approach of the previous slide.

Creating the XHR Object – Use of External File

- Note that **xhr** is just our *chosen* name for the XHR object that communicates between client and server; any name could be chosen, as in later examples
- We define the object via a call to an external function, that handles browser differences (see later slide)
- The function **createRequest**, which returns a valid XHR object, is defined in the JavaScript file **xhr.js**. That file is indicated as an included script in our main HTML file.

XHR Object creation

```
// file xhr.js

// return a valid XHR object

function createRequest() {
    var xhrObj = false;

    if (window.XMLHttpRequest) {
        xhrObj = new XMLHttpRequest();
    }

    else if (window.ActiveXObject) {
        xhrObj = new ActiveXObject("Microsoft.XMLHTTP");
    }

    return xhrObj;
} // end function createRequest()
```

Check for native XHR object first, as most contemporary browsers have this. Use the native object in IE where possible.

Older versions of IE do not have the native XHR object, so we then create an appropriate ActiveXObject that has similar functionality

Defining the XHR Object – Browser Differences

- Our implementation is very simple, and just covers essential differences between Firefox and older versions of IE. For IE7+, it will use the native XMLHttpRequest object available rather than the ActiveX object.
- A more sophisticated implementation is needed to give wider browser availability.
- If neither condition in the code is satisfied, then no XHR object is created, and the function returns false.
- The function call does not fail, but the possibility of false being returned should be handled in the calling code.

Server Script : data.php

```
<!--file data.php -->
<?php
    // get name and password passed from client
    $name = $_GET['name'];
    $pwd = $_GET['pwd'];
    sleep(10); // simulate delay at server to make async obvious
    // write back the password concatenated to end of the name
    echo ($name." : ".$pwd)
?>
```

What Happens?

- We start by loading “simpleajax.htm” in the browser
- When this loads, the JavaScript files in the header load
 - **xhr.js** just makes available the function createRequest() that creates an XHR object
 - **simpleajax.js** is executable code, and when it is loaded
 - The XHR object named xhr is created via a call to function createRequest() {**xhr.js is loaded first to enable this**}
 - The function getData() is made available – it does not execute until it is called
- The body of the HTML now loads

What Happens?

- This basically displays the form for the user interface, and prepares the location for the returned data (an HTML div). Nothing happens – it is waiting for us to enter the data fields and press the button
- When we press the button, the registered “onclick” event causes the JavaScript function `getData()` to be called
- This function executes; it sets up variable “place”, opens up the connection of xhr to the server, specifying that the “GET” protocol will be used, specifies the PHP file on the server that will load to xhr, specifies that processing is to be asynchronous (via the use of “true” as the third parameter) - and defines the (anonymous) call-back function that will execute when the ready state of xhr changes

What Happens?

- Next, start the client – server communication via xhr by “send” request; this sends the data as part of the URL in the GET protocol.
- Then it activates the execution of the PHP script on the server, which in this case simply takes the data passed as parameters, concatenates them, sleeps for 10 seconds (to simulate heavy load on the server – don’t do this in a real system!!) and echos the result.
- This essentially writes the resulting string to xhr’s **responseText** property.
- Whilst the user waits for the response (s)he can interact with the system.

What Happens?

- As the communication continues, the server updates properties of xhr, keeping xhr aware of the state of the communication; every time the state changes, the **onreadystatechange** event is fired on xhr, and the anonymous call-back function (mentioned on the previous slide) executes – each time it executes we get an alert box that shows the new value of the **readyState** property of xhr
- It is here to show how the event fires several times as the state changes, before finally indicating that the communication has ended)

What Happens?

- The conditional code in the call-back function checks the `readyState` and `status` properties of `xhr`
- If the former is 4, and the latter is 200 (ie, the server has completed sending its response, and the status of `xhr` is “OK”), then the `innerHTML` property of the location represented by the variable `place` is assigned the value of the `responseText` property of `xhr` (which is the concatenated string sent from the server)
- Note that the state of `xhr` changes several times and consequently the call-back function executes several times prior to the completion of the communication.