AJAX (Asynchronous Javascript and XML)

An Introduction

- AJAX allows web pages to be updated asynchronously by exchanging data with a web server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

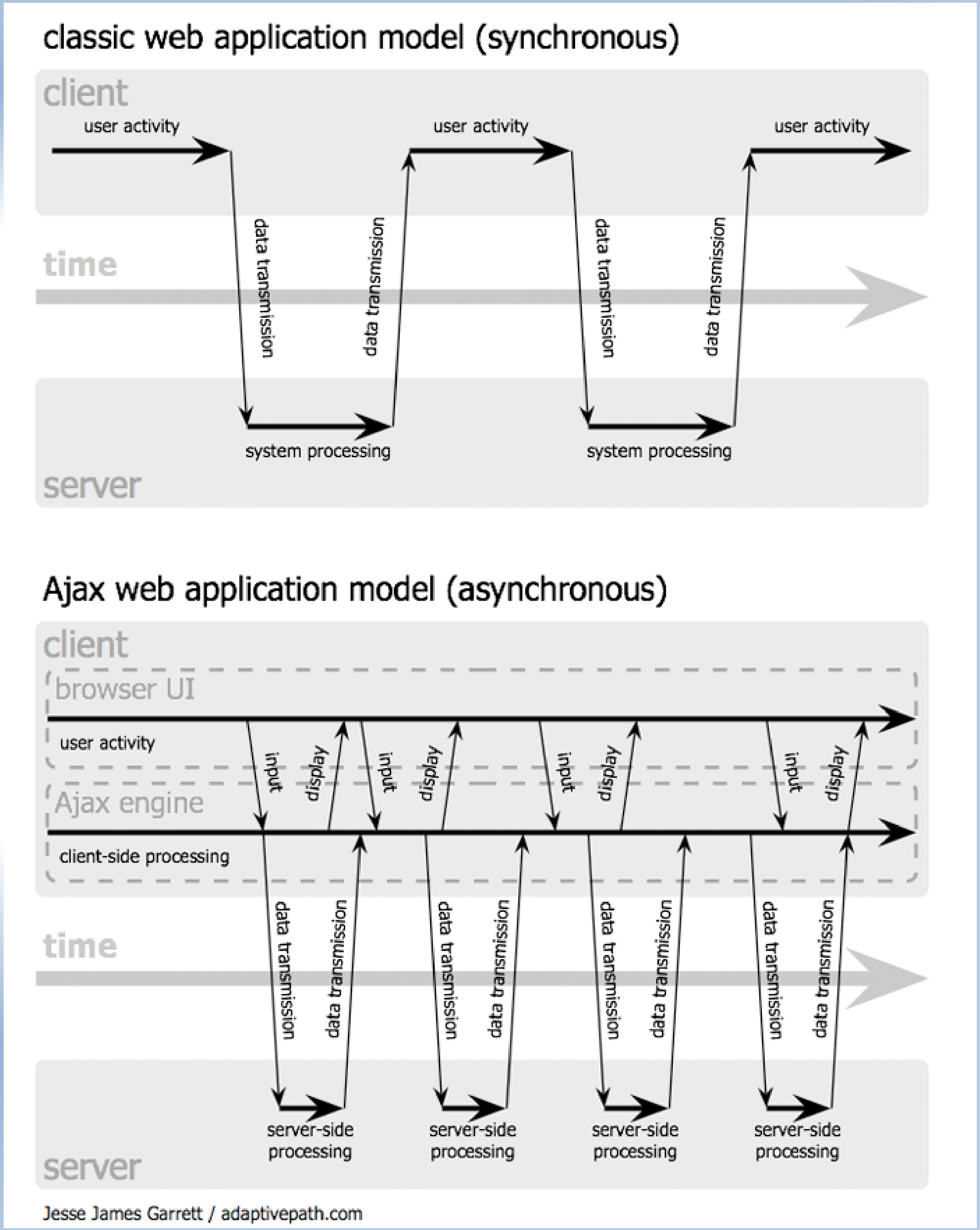
AJAX is based on internet standards, and uses a combination of:
•XMLHttpRequest object (to retrieve data from a web server)
•JavaScript/DOM (to display/use the data)

The Good things:

Update a web page without reloading the page

Request and receive data from a server - after the page has loaded

AJAX in Operation



Source: http://www.adaptivepath.com/uploads/archive/images/publications/essays/ajax-fig2.

- One thing to notice about AJAX in use currently, is that it rarely involves XML

Current use is more likely to involve JSON

  - It is more compact
  - It is easier for humans to read)
  - It is a simpler format, so processing is faster

JSON:

- A lightweight text based data-interchange format
- language independent
- A subset of the object literal notation of JavaScript (or ECMA-262).

http://json.org/

JSON is built on two structures

1. A collection of name/value pairs.
   – In various languages, this is realized as an ***object*** in JavaScript.

2. An ordered list of values.
   – this is realized as an ***array*** in JavaScript.
   – e.g.: An array of four integers:
   – **`[ 100,56,28,4847]`**

```
var family = [{
"name" : “Matthew",
"age" : "24",
"gender" : "male"
},
{
"name" : “Elizabeth",
"age" : "21",

}];
```

- **The XMLHttpRequest Object**

- The XMLHttpRequest object is used to exchange data with a server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.

AJAX Now

- The original idea was to allow web pages to be augmented according to user-interactions
  - AJAX is now a core technology for updating a browser’s document object model in applications

```
var xhttp;
if (window.XMLHttpRequest) {
   xhttp = new XMLHttpRequest();
   } else {
   ..error code if request object not supported
}
```

- To send a request to a server, we use the open() and send() methods of the XMLHttpRequest object:

Example:

- xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();

- A simple GET request:

- xhttp.open("GET", "demo_get.asp?mycar = lada", true);
  xhttp.send();

Post:

```
xhttp.open("POST", "demo_post.asp", true);
xhttp.send(“mycar = lada”);
```

- **The url - A File On a Server**

- xhttp.open("GET", **"ajax_test.asp",** true);

- **Asynchronous - True or False?**

- xhttp.open("GET", "ajax_test.asp", **true**);

Server Response

To get the response from a server, use the responseText or responseXML property of the XMLHttpRequest object.

Example

```
document.getElementById("demo").innerHTML =
xhttp.responseText;
```

- **Async = true**

- When using async = true, specify a function to execute when the response is ready in the onreadystatechange event:

- **Example**

-
```
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    document.getElementById("reply").innerHTML = this.responseText;
  }
};
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Note: if you use async=false, do NOT write an onreadystatechange function - just put the code after the send() statement:

The onreadystatechange event

onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request **4: request finished and response is ready**
status	**200: "OK"** 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The onreadystatechange function is called every time the readyState changes.

When readyState is 4 and status is 200, the response is ready:

Example

```
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
```

- **Using a Callback Function**
- A callback function is a function passed as a parameter to another function.
- The function call should contain the URL and what to do on onreadystatechange (which is probably different for each call):

AJAX at its simplest

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>Simple AJAX Demo</title>
</head>
<body>
    <h1 id="title"></h1>
    <script src="ajax.js”></script>
</body>
</html>
```

- AJAX Needs
  - A Target element that will be updated
  - Script to download and update the target element
  - The XHR object (shorthand for XMLHttpRequest)

```
var xhr = new XMLHttpRequest(),
    readyStates = [
        "0: request not initialized",
        "1: server connection established",
        "2: request received",
        "3: processing request",
        "4: request finished and response ready"
    ];
xhr.open("GET", "title.txt", true);
xhr.onreadystatechange = function() {
    console.log(readyStates[xhr.readyState]);
    if(xhr.readyState == 4 && xhr.status == 200) {
        document.getElementById("title").innerText
                                 = xhr.responseText;
    }
}
xhr.send();
```

- XHR Returns
  - .readyState
  - .status
  - .responseText,
    .responseXML
  - Other info.

AJAX and jQuery Mobile

- As the example shows, AJAX works with URLs to get data
- The data can be local or remote
  - e.g. the contents of a text file on the server
  - e.g. data from online databases (in the right format)

- The call can even be made to get data from within the file that makes it
  - e.g. in jQM, the readystatechange function is used to show/hide DOM elements, play animations, attach CSS styles to elements etc.
    - AJAX updates the document elements AND keeps the browser history in line

AJAX in jQM

- Since jQM apps must also include jQuery
  - The built-in AJAX mechanism is easy to get to
  - AJAX code tends to be simpler to set up and provides more useful information
    - See http://demos.jquerymobile.com/1.3.0-rc.1/docs/demos/widgets/ajax-nav/

Needed for cross-origin

```
$.ajax({
   datatype: "jsonp",
   url: ratesURL + symbol,
   success: function(data) {
      rateList = data;
      doUpdates();
   },
   error: function(err) {
      alert("Error: " + err.message);
   }
});
```

Processing the AJAX response

```
function doGetRequest() {
  // The parameters for this are important.
  $.ajax({
    type: "GET",                          // The HTTP operation
    url: "http://<some-service-url>",     // The service URL
    jsonpCallback: 'handleResults',       // Function to call with results.
    contentType: "application/json",      // MIME type function expects...
    dataType: 'jsonp',                    // ...and the type of data expected
    error: function(e) {                  // Do this if it failed
      confirm("Error", e.message);
    }
  });
```

```
function handleResults(messages) {
  var i, list = "";
  for(i=0; i < messages.length; i += 1) {
    list += formatMessage(messages[i]);
  }
  displayResults(list);
}
```

- handleResults() simply has to deal with JSON data
- In this case, a list of messages on a messageboard
- The callback function gets results passed in a parameter

Providers of JSONP

- For all this to work, you need to access a site that returns JSONP – some do so directly…
  - Twitter, Facebook and various weather, finance, entertainment sites etc. do this
  - e.g.
    - https://dev.twitter.com/docs/api/1/get/statuses/user_timeline
    - http://www.footytube.com/openfooty/service.php?package=League&method=getResults
    - http://www.programmableweb.com/api/met-office-datapoint
    - http://currencyfeed.com/
    - http://code.google.com/p/yahoo-finance-managed/wiki/YahooFinanceAPIs
    - http://www.apple.com/itunes/affiliates/resources/blog/introduction

JSONP Data Sources

- For web-apps, we MUST use JSONP (unless it is our own server)
  - Not as big a limitation as you might think
    - The Met Office – weather data from the horses mouth
      - www.metoffice.gov.uk
    - Yahoo – a huge range of services
      - https://developer.yahoo.com/yql/console/
    - Google Apps – return data from Google Docs
      - https://developers.google.com/apps-script/guides/content
    - Also twitter, facebook, news sites
- These will all deliver JSON data to a client
  - Add a callback parameter, and they'll deliver JSONP
    - e.g. https://itunes.apple.com/gb/rss/topmovies/limit=10/json?callback=func