Client Scripting - Ajax
 Ajax

• What is Ajax?
  – Shorthand for Asynchronous JavaScript and XML
  – Is a technique of designing web apps / single-page apps
  – Makes use of
    • scripted http requests
    • (aka web remoting)
  – In essence:
    • Without replacing the current page
    • Make a new request to a server
    • Process the response
    • Update the display

• Ajax is the core of all single-page apps today:
  – Gmail
  – Piazza
  – Tumblr
  – Many many others ...
Some Ajax Patterns

- Dynamically update content from remote source
- Friendly interactive interface to remote services
- Responsive first-person interaction
  - Instead of click-and-wait
- Collaborative interaction
- Streaming within a web-page context
- Submission throttling
  - E.g. Google docs (saves)
- Partial completion
  - E.g. Google Suggest
Very simple example

• Demonstrate xhr-simple.html

• Clicking the button
  – fetches text from a text file
  – adds it to the DOM
• Key to the Ajax style communication is communication between a browser and a server within the context of a page
  – i.e. without a new page load
• XMLHttpRequest is the primary means of facilitating this communication
• All modern browsers have a built-in XMLHttpRequest object.
XMLHttpRequest basics

1. Get a *new XMLHttpRequest* object
2. *Open* it
   - Non-blocking (or blocking (less common))
3. *Send* a request
4. When complete, process the response
1. Get a new XMLHttpRequest object

- var xhr = new XMLHttpRequest();
2. Open

E.g. xhr.open("GET", "data.txt", true);

Parameters:

• 1st: http request method (text string)
  – i.e. get, post (most often)
  – Also head, put, delete, options

• 2nd: URL of HTTP request (text string)
  – Is relative to the URL of the document containing the script that is calling open.
  – Typically restricted to be on the same protocol://server:port

• 3rd: boolean - is the request asynchronous
  – True means a send will not block (asynchronous)
  – AVOID: False means a send will block (synchronous)
Blocking (synchronous) requests

• If synchronous
  – Send...
    • Will block until response fully received
  – Check if response.status == 200  // OK
  – Use results

• Synchronous should be avoided in most situations
  – The browser can feel unresponsive and/or hang
    • Later releases don't really "block" to avoid this.
  – Some warn that this can lead to memory leaks.
    • because of unanticipated synchronization problems

• We will only deal with non-blocking (asynchronous) xmlhttprequests.
Optional: add headers to the request

• You can add headers to the request
  – E.g. xhr.setRequestHeader("Content-Type", "text/plain");

• Caveats:
  – Add the same header twice, and it is not replaced, but added.
  – Cookies are handled automatically
  – Some other headers are also added automatically, for example:
    • referer
    • user-agent
  – Consequently, you cannot alter these headers in code.
3. Send – initiates request to server

• GET, HEAD, PUT, DELETE, OPTIONS
  – Put parameters in the open URL query string
  – Then send(null)
  – E.g.:
    open("GET","http://localhost/foo/test?param1=x&param2=y", true);
    send(null);

• POST
  – Put parameters as arguments to send
  – E.g.:
    open("POST","http://localhost/foo/test", true);
    send("param1=x&param2=y");

• Send() initiates communication with the server.
4. Handle response

• Define a callback function to process the response from the server.

• The browser calls an event handler for each change in the state of the XMLHttpRequest object:

• Therefore you must define a callback function

  `xhr.onreadystatechange = function() { ...`
XMLHttpRequest readyState

• xhr.readyState values:
  0. Object has been constructed
  1. Open method successful
     • headers can now be set
     • send can now be done
  2. Response headers have been received from the server
  3. Response body is being received (in progress)
  4. Response complete (or something went wrong)

• The onreadystatechange event fires each time the value of readyState changes.
xhr.onreadystatechange = function() {
  if (xhr.readyState == 4) {
    if (xhr.status == 200) {
      // Process the response
    } else {
      document.getElementById("responseArea").innerHTML = "Error code " + xhr.status;
    }
  }
}
Use the results

• XML
  – *Iff* the response is valid XML, it can be treated as such
  – `responseXML` will be a DOM document object
    • e.g. xhr.responseXML

• JSON
  – Can be parsed into a JavaScript object using `JSON.parse()`

• Text
  – The results can be treated as String
  – `responseText`
    • e.g. xhr.responseText
    • Can be manipulated as any String object
What is JSON?

• JavaScript Object Notation
• Lightweight data format
• Essentially, JavaScript literal format
• For details, see http://json.org
Demo

- Processing XML
  - xhr-simpleXML.html & data.xml
  - xhr-simpleXML2.html & data2.xml
- Processing JSON
  - xhr-simpleJSON.html & data.json
Most of the time you will know the content
This demonstrates how to deal with each kind of content.
And what to do if you are not sure.

// Find the content type of the response
var type = xhr.getResponseHeader("Content-Type");
if (type.indexOf("xml") !== -1 && xhr.responseXML) {
    // if valid XML, responseXML will be a DOM (and not false)
    // process xhr.responseXML, e.g. xhr.responseXML.getElementsByTagName("book")
} else if (type === "application/json") {
    // use JSON.parse(xhr.responseText) to turn the response into a JavaScript object
} else {
    // response is text, so process xhr.responseText as a String
}
Caveat per file: // and xmlhttprequest

• xmlhttprequest is a browser-level API for http requests and replies
• In other words, it relies on http protocol behavior
• xmlhttprequest does not fully work on pages accessed with a file: // URL
• Therefore need to access pages from a web server
  • I.e. OpenShift