

INT222 - Internet Fundamentals

Week 12: AJAX and BOM

Agenda

- AJAX
 - Simulating a call to a Web service
- JSON
- Browser Object Model

What is AJAX

- AJAX stands for **A**synchronous **J**ava**S**cript and **X**ML.
- Coined in 2005 by Jesse James Garrett.
- Not a technology , but a “new” way to use existing technologies.
- AJAX is a group of interrelated web techniques used on the client-side for creating fast and dynamic web pages.

Technologies used in AJAX

- HTML
- CSS
- JavaScript
- The Document Object Model (DOM)
- XML/JSON, XSLT
- XMLHttpRequest object

AJAX Examples

- Google Maps

<http://maps.google.com/>

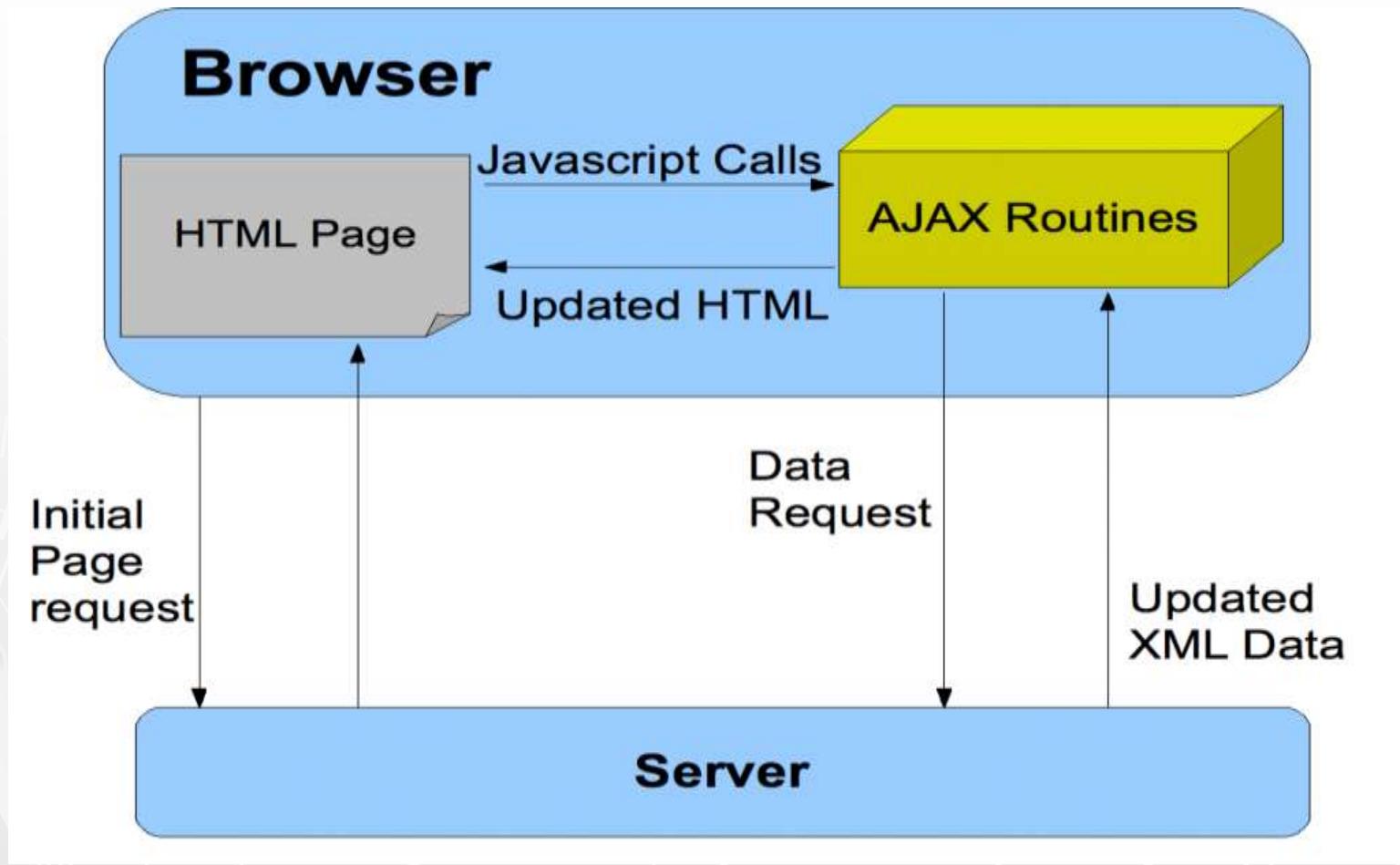
- Google Suggest

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

How does it works?

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

The AJAX model



Comparing with classic apps

- Conventional web application transmit information to and from the sever using synchronous requests. This means you fill out a form, hit submit, and get directed to a new page with new information from the server.

Packaging data in AJAX model

- XML was originally used as the format in **AJAX**.
- JSON is used more than XML nowadays .
 - Advantages to use JSON: being lighter and a part of JavaScript.
- Any format, including plain text, can be used.

Features

- Make requests to the server without reloading the page
- Receive and work with data from the server
- AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background
- Intuitive and natural user interaction. No clicking required only Mouse movement is a sufficient event trigger.
- Data-driven as opposed to page-driven

XMLHttpRequest object

- It is the most important part of AJAX.
- It is a JavaScript object
 - Designed by MS and adopted by Mozilla, Apple,
...
- It provides an easy way to retrieve data from a URL without having to do a full page refresh.
- Creation:

```
var myRequest = new XMLHttpRequest();
```

XMLHttpRequest Methods

- `abort()`
- `getResponseHeader()`
- `open() // Initializes a request.`
- `overrideMimeType()`
- `send() // Sends the request.`

➤ ...

XMLHttpRequest properties

- `onreadystatechange`
- `readyState`
- `responseText`
- `responseType` // set to change the response type
- `responseXML`

Writing AJAX

Step 1 – makes an HTTP request object

```
// creating a cross-browser instance
var httpRequest;
if (window.XMLHttpRequest) { // Mozilla, Safari, ...
    httpRequest = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE 8 and older
    httpRequest = new
    ActiveXObject("Microsoft.XMLHTTP");
}
```

Writing AJAX

- **Step 2 – register a request listener**

```
httpRequest.onreadystatechange = reqListener;
```

```
function reqListener () {  
    // process the server response  
};
```

- **Or:**

```
httpRequest.onreadystatechange = function(){  
    // process the server response  
};
```

Writing AJAX

➤ **Step 3 - make the request**

```
// Specifies the type of request
```

```
httpRequest.open('GET',
```

```
  'http://www.example.org/some.file', true);
```

```
// Sends the request off to the server.
```

```
httpRequest.send(null);
```

Parameters of open() and send()

- Parameters of the open(*method, url, async*) method:
 - 1st para: HTTP request method - GET, POST, ...
 - 2nd para: the URL of the page this requesting
 - 3rd para: sets whether the request is asynchronous
- Parameters of the send(*string*) method:
 - The "*string*" para: Only used for POST requests.
 - ▶ any data you want to send to the server if using POST method. This can be as a query string, like:

```
name=value&anothername)+"+encodeURIComponent(myVar)+"&so=on"
```

Writing AJAX

➤ **Step 4 – handling the server response**

```
// check for the state of response
if (httpRequest.readyState === 4) {
    // everything is good, the response is received
} else {
    // still not ready
}


- the readyState values:
  - ▶ 0 (uninitialized), 1 (loading), 2 (loaded), 3 (interactive), 4 (complete)

```

Writing AJAX

➤ **Step 4 – handling the server response (cont')**

```
// check is the response code of the HTTP server
response
if (httpRequest.status === 200) {
    // perfect!
} else {
    // there was a problem with the request,
    // for example the response may contain a 404 (Not
    Found)
    // or 500 (Internal Server Error) response code
}
```

Writing AJAX

- Set the button to start

```
<button type="button" onclick="makeRequest();">  
  Make a request</button>
```

Working with the XML response

- Suppose server response is in XML format:

```
<?xml version="1.0" ?>  
<root>  
    I'm a test.  
</root>
```

- Parsing XML data

```
var xmldoc = httpRequest.responseXML;  
var root_node = xmldoc.getElementsByTagName('root').item(0);  
alert(root_node.firstChild.data);
```

Working with the JSON response

- Suppose server response is in JSON format:
{"name": "Kevin", "age": 22 }

- Parsing JSON data:

```
// Javascript function JSON.parse to parse JSON data
var jsonObj = http_request.responseText;
var jsObj = JSON.parse(jsonObj);
var name = jsObj.name;
var age = jsObj.age;
```

About JSON



- JSON stands for **JavaScript Object Notation**.
 - specified by Douglas Crockford.
 - a lightweight text based data-interchange format.
 - "self-describing" and easy to understand
 - smaller than XML, and faster and easier to parse
- JSON parsers and JSON libraries exists for many different programming languages.
 - e.g. C, C++, Java, Python, Perl, PHP etc.
- JSON filename extension is **.json**
- JSON Internet Media type is **application/json**

JSON Structures

- JSON is built on two structures:

- **JSON object**: a collection of name/value pairs.

```
{ "firstName": "John",  
  "lastName": "Smith", "age": 25,  
  "address": { "street": "21 2nd street",  
               "city": "North York",  
               "province": "ON",  
               "postalCode": "M2M6T6" }  
}
```

- **JSON array**: an ordered list of values.

```
[  
  {"name" : "Kevin", "age" : 22, "gender" : "m"},  
  {"name" : "Kate", "age" : 22, "gender" : "f"},  
  {"name" : "Steven", "age" : 25, "gender" : "m"},  
  {"name" : "Bill", "age" : 22, "gender" : "m"}  
]
```

JSON Objects vs JavaScript Objects

- JSON and JavaScript use the same syntax to describe data objects, including Arrays.
- But JSON objects are text-based or "string"s.
- The properties of JSON objects have to be quoted.
e.g.
`{ "name": "Kevin", "age": 22 }`

JSON Objects vs JavaScript Objects

- Conversion between JSON and JavaScript objects – using JS built-in object: **JSON**

- Serializing JavaScript object (converting JavaScript object to JSON object) :

```
var jsObject = { "language" : "Java", "course" : "JAC444" };  
var JSONString = JSON.stringify(object1);
```

- Parsing JSON string (converting JSON object to JavaScript object) :

```
var JSONString2 = '{ "language" : "C++", "course" : "OPP344" }';  
var jsObject2 = JSON.parse(JSONString2);
```

JSON vs XML

➤ A JSON object

```
{ "menu": { "id": "file",
             "value": "File",
             "popup": {
               "menuitem": [
                 {"value": "New", "onclick": "CreateDoc()"},
                 {"value": "Open", "onclick": "OpenDoc()"},
                 {"value": "Close", "onclick": "CloseDoc()"} ] } } }
```

➤ Equivalent XML document

```
<?xml version="1.0" ?>
<root>
  <menu id="file" value="File">
    <popup>
      <menuitem value="New" onclick="CreateDoc()" />
      <menuitem value="Open" onclick="OpenDoc()" />
      <menuitem value="Close" onclick="CloseDoc()" />
    </popup>
  </menu>
</root>
```

JavaScript: simulating AJAX calls to Web Services

- Example 1: calling a web service with **JSON object** response
 - Code:
<https://zenit.senecac.on.ca/~wei.song/int222/ajax/ajaxjson.html>
- Example 2: calling a web service with **JSON array** response
 - Code:
<https://zenit.senecac.on.ca/~wei.song/int222/ajax/ajaxjson2.html>

Exercise

- Create an HTML5 page that loads data with an AJAX call to a web service.
- The web service is simulated by a static JSON file:
<https://zenit.senecac.on.ca/~wei.song/int222/json/student.json>
- You have to HTML page to the ZENIT server to make the AJAX code work.
 - ATTN: usually, Cross-Domain AJAX call is not allowed for security reason.
- The HTML page is showed at right.

Info of an ICT Student

Name	Kevin
Age	22
Program	CPD
Course 1	INT222
Course 2	DBS201
Course 3	OOP244

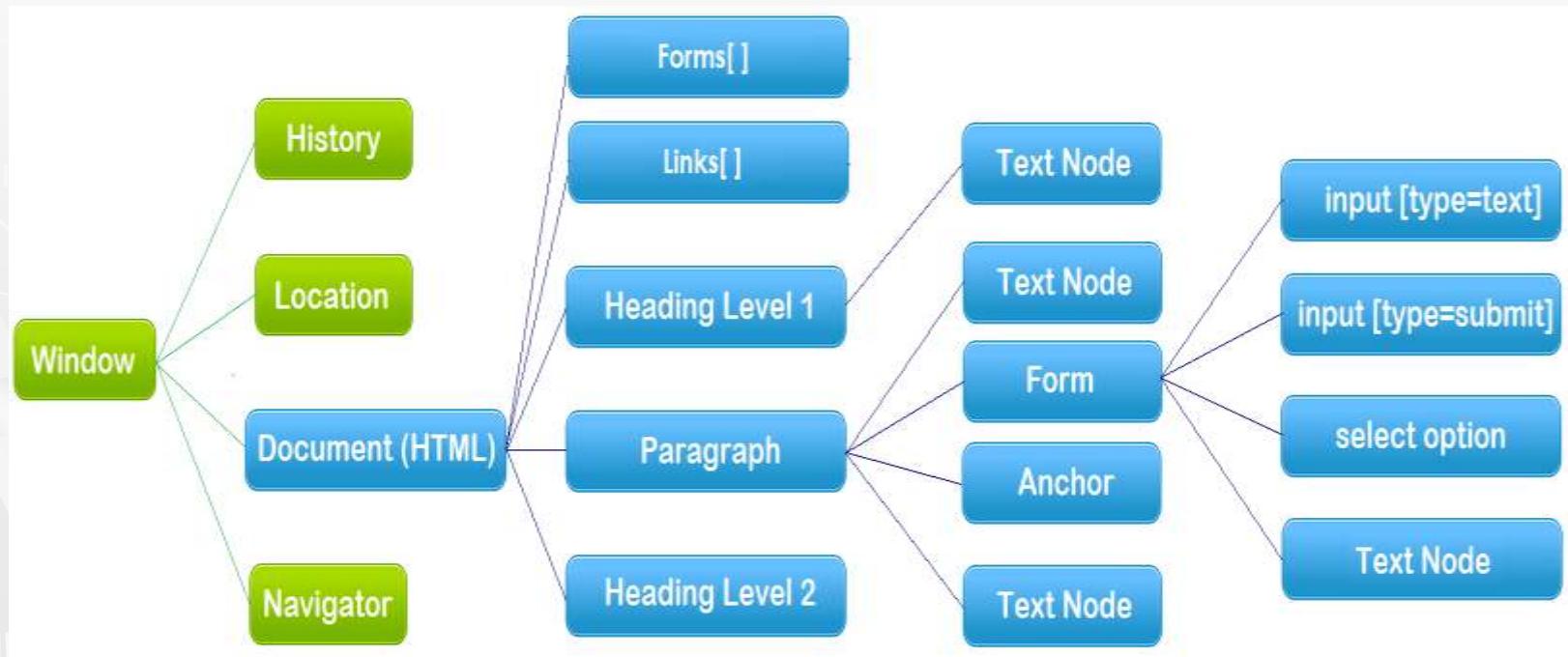
Make a request

Browser Object Model

(BOM)

The Browser Object Model (BOM)

- Browser Object Model (BOM) is the collection of objects which define the browser window and its contents within the browser that can be accessed by JavaScript.
- The hierarchy of browser objects



The Browser Object Model (BOM)

- Many people mix the concepts of Browser Object Model (BOM) with Document Object Model (DOM).
 - Strictly, DOM is subset of BOM. (See the blue part of the diagram in last page, and we'll cover or in future class).
- There are no official standards for the Browser Object Model (BOM).
 - But, the DOM is a W3C (World Wide Web Consortium) standard.
- The top level of the BOM hierarchy is the window object, which contains all other objects:
 - location object,
 - history object,
 - document object,
 - navigator object

Browser Object - window

- An open window in a browser.
- The top level object in the **window** Object hierarchy.
- All other objects derive from the **window**.
 - Global variables are properties of the **window** object.
 - Global functions are methods of the **window** object.

Browser Object - window

- Property: **document**
 - Syntax: **window.document** or **document**
- Property: **location**
 - Syntax: **window.location** or **location**
- Property: **history**
 - Syntax: **window.history** or **history**
- Property: **navigator**
 - Syntax: **window.navigator** or **navigator**

BOM – Window Object

➤ Method – **open()**

- Opens a new browser window
- Syntax:

```
var winObjRef = window.open(strUrl, strWindowName,  
[strWindowFeatures]);
```

- **strUrl**, optional. Open the page in the URL. If no URL (empty string ""), a new window with about:blank is opened
- **strWindowName**, Optional. Specifies the target attribute or the name of the window.
 - **_blank** - URL is loaded into a new window. default
 - **_parent** - URL is loaded into the parent frame
 - **_self** - URL replaces the current page
 - **_top** - URL replaces any framesets that may be loaded
 - **name** - The name of the window

BOM – Window Object

- **[strWindowFeatures]**, an optional parameter listing the features (size, position, scrollbars, etc.) of the new window as a string.
 - The string must not contain any whitespace, and each feature name and its value must be separated by a comma.
 - See Position and size features details from [this link](#).
 - e.g.
"toolbar=no,menubar=no,scrollbars=yes,resizable=yes,directories=no,status=no,location=no,width=450,height=550"

BOM – Window Object

- Example: open and write a new page

```
var myWindow = window.open("", 'winName', 'width=400,height=200,  
scrollbars = yes');
```

```
myWindow.document.write("<p> write to new window in <span style  
= 'color: blue;'> blue </span>. </p>");
```

```
myWindow.document.close(); //closes the output stream to the document.  
myWindow.theWindow.focus();
```

```
// myWindow.close();
```

BOM – Window Object

- Examples: open an existing web pages

```
window.open('http://www.google.com');
```

```
myWindow =  
  window.open('http://www.google.com','MsgWindow','width=400,  
  height=200');
```

```
// myWindow.focus();  
// myWindow.blur();  
// myWindow.close();
```

[js-window-open.html](#)

Browser Object - Window

- Method: **close()**
 - Close the current window.
- Property: **name**
 - Sets or returns the name of the window.
- Method: **blur()**
 - removes focus from the current window(send the window to the background).
- Method: **focus()**
 - sets focus to the current window.
- Method: **moveBy()**
 - moves a window a specified number of pixels relative to its current coordinates.
- Method: **moveTo(x,y)**
 - moves a window's left and top edge to the specified coordinates.

Browser Object - Window

➤ Global built-in functions

- Alert box: `alert("An alert box!");`
- Confirm box: `var ans = confirm("Are u ok?");`
- Prompt box: `var namw = prompt("Enter your name", "Bill");`
- `isNaN()`, `isFinite()`
- `parseInt()`, `parseFloat()`, `Number()`
- `eval()`

➤ Global properties

- `Infinity`, `NaN`, `undefined`

➤ JavaScript built-in objects

- `String`, `Array`, `Date`, `Number`, ...

➤ Custom functions, global variables

Browser Object - Navigator

- The navigator object contains information about the browser.

Property	Description
appCodeName	Returns the code name of the browser
appName	Returns the name of the browser
appVersion	Returns the version information of the browser
cookieEnabled	Determines whether cookies are enabled in the browser
platform	Returns for which platform the browser is compiled
userAgent	Returns the user-agent header sent by the browser to the server
geolocation	Returns a Geolocation object that can be used to locate the user's position

□ js-navigator.html

Browser Object - History

- The History object interface allows to manipulate the browser session history
- The history object contains the URLs visited by the user (within a browser window).
- The `window.history` read-only property returns a reference to the History object.

Browser Object - History

➤ Property: **length**

- Returns the number of URLs in the history list.
- Syntax: **history.length**
- Note: Internet Explorer and Opera start at 0, while Firefox, Chrome, and Safari start at 1.

➤ Method: **go()**

- Loads a specific URL from the history list.
- Syntax: **history.go(number|url)**
- The parameter can either be:
 - a **number** which goes to the URL within the specific position (-1 goes back one page, 1 goes forward one page)
 - a **string** that must be a partial or full URL, and the function will go to the first URL that matches the string.

Browser Object - History

- Method: **back()**
 - Syntax: **history.back()**
 - The back() method loads the previous URL in the history list.
 - This is the same as clicking the Back button or **history.go(-1)**
- Method: **forward()**
 - Syntax: **history.forward()**
 - The forward() method loads the next URL in the history list.
 - This is the same as clicking the Forward button or **history.go(1)**.

[js-history.html](#)

Browser Object - location

- The location object contains information about the current URL.
- The location object is part of the window object and is accessed through the window.location property.

Property	Description
----------	-------------

<u>host</u>	Returns the hostname and port of a URL
-------------	--

<u>hostname</u>	Returns the hostname of a URL, the server name, subdomain and domain name of a location
-----------------	---

<u>href</u>	Returns the entire URL, a string which contains the location URL
-------------	--

<u>port</u>	Returns the port number the server uses for a URL
-------------	---

<u>protocol</u>	Returns the protocol of a URL, method of access to the URL (http, https, ftp, etc.)
-----------------	---

Browser Object - location

- Redirect web page (in JS)

```
<script>
```

```
location.href='http://www.senecacollege.ca/';
```

```
// or
```

```
// location='https://open.senecac.on.ca/cms/';
```

```
</script>
```

js-location.html

Resource & Reference

- **MDN: AJAX - Getting_Started**
 - https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started
- **JavaScript Object Notation (JSON)**
- **Browser Object Model - Wikipedia, the free encyclopedia**

Thank You!