

# AJAX

- Ajax motivation
- The basic Ajax process
- The need for anonymous functions
- Using dynamic content and JSP
- Using dynamic content and servlets
- Displaying HTML results

# What is AJAX?

- What is in the name ?

- **AJAX** stands for :

**Asynchronous** JavaScript and XML

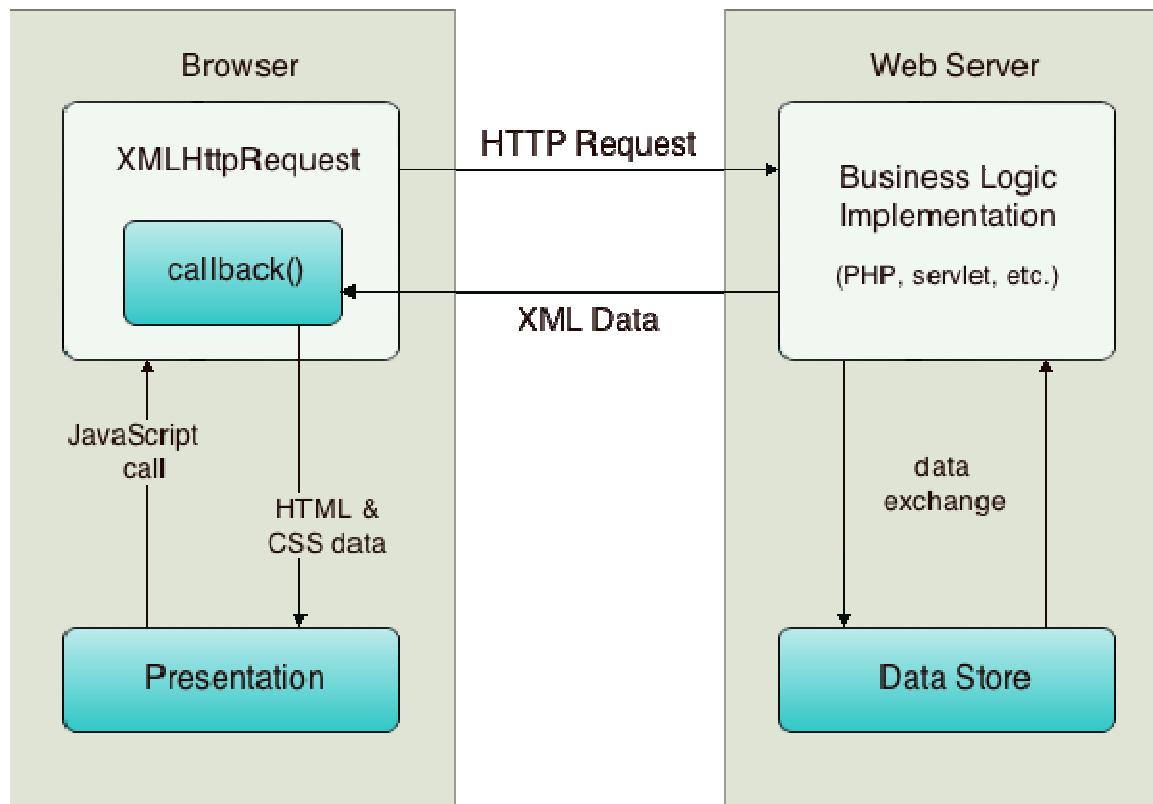
# What is Ajax?

- In essence, Ajax is an efficient way for a web application to handle user interactions with a web page
  - *a way that reduces the need to do a page refresh or full page reload for every user interaction.*
- Ajax interactions are handled asynchronously in the background. As this happens, a user can continue working with the page.
- Ajax interactions are initiated by JavaScript code. When the Ajax interaction is complete, JavaScript updates the HTML source of the page. **The changes are made immediately without requiring a page refresh.**

# What is Ajax?

- Ajax interactions can be used to do things such as :
  1. Validate form entries (while the user is entering them) using server-side logic,
  2. Retrieve detailed data from the server,
  3. Dynamically update data on a page, and
  4. Submit partial forms from the page.

# What is Ajax?



# What is Ajax?

- The process flow of the diagram can be described by the following steps:
  1. The user triggers an event, for example by releasing a key when typing in a name. This results in a JavaScript call to a function that initializes an XMLHttpRequest object.
  2. The XMLHttpRequest object is configured with a request parameter that includes the ID of the component that triggered the event, and any value that the user entered. The XMLHttpRequest object then makes an asynchronous request to the web server.
  3. On the web server, an object such as a servlet or listener handles the request. Data is retrieved from the data store, and a response is prepared containing the data in the form of an XML document.
  4. Finally, the XMLHttpRequest object receives the XML data using a callback function, processes it, and updates the HTML DOM (Document Object Model) to display the page containing the new data.

# Why Ajax?

- Solve three key problems of Web apps
  - Coarse-grained updates
  - Synchronous: you are frozen while waiting for result
  - Extremely limited options for widgets (GUI elements)

# What is XMLHttpRequest?

The screenshot shows a Mozilla Firefox browser window with the title bar "XMLHttpRequest - Mozilla Firefox". The address bar contains the URL "www.w3.org/TR/XMLHttpRequest/". The page content is the W3C XMLHttpRequest Working Draft from December 6, 2012. It features the W3C logo and the title "XMLHttpRequest". Below the title, it says "W3C Working Draft 6 December 2012". The page lists several URLs for different versions of the XMLHttpRequest specification, starting from the "This Version" at <http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/>. It also includes links for the "Latest Version" (<http://www.w3.org/TR/XMLHttpRequest/>), "Latest Editor Draft" (<http://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html>), and a list of "Previous Versions" with their respective URLs. At the bottom, it lists the "Editor" (Julian Aubourg, Jungkee Song, Hallvard R. M. Steen) and the "Previous Editor". The browser interface includes standard toolbar icons and a McAfee security badge.

XMLHttpRequest

W3C Working Draft 6 December 2012

This Version:  
<http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/>

Latest Version:  
<http://www.w3.org/TR/XMLHttpRequest/>

Latest Editor Draft:  
<http://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html>

Previous Versions:

<http://www.w3.org/TR/2012/WD-XMLHttpRequest-20120117/>  
<http://www.w3.org/TR/2011/WD-XMLHttpRequest2-20110816/>  
<http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/>  
<http://www.w3.org/TR/2009/WD-XMLHttpRequest2-20090820/>  
<http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080930/>  
<http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080225/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070618/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070227/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060619/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>

Editor:  
[Julian Aubourg \(Creative Area\) <j@ubourg.net>](#)  
[송정기 \(Jungkee Song\) \(Samsung Electronics Co., Ltd.\) <jungkee.song@samsung.com>](#)  
[Hallvard R. M. Steen \(Opera Software ASA\) <hallvard@opera.com>](#)

Previous Editor:

# Origin of XMLHttpRequest?

The screenshot shows a Mozilla Firefox browser window displaying the W3C XMLHttpRequest specification. The title bar reads "XMLHttpRequest - Mozilla Firefox". The address bar shows the URL "www.w3.org/TR/XMLHttpRequest/#introduction". The left sidebar is labeled "V3C Working Draft" and contains links for "4.8 Events summary", "5 Interface FormData", "6 data: URLs and HTTP", "References", and "Acknowledgments". The main content area starts with "1 Introduction" and includes a note: "This section is non-normative." A yellow box highlights the first paragraph: "The XMLHttpRequest object is the ECMAScript HTTP API. [ECMASCIPT]".

The name or the object is XMLHttpRequest for compatibility with the web, though each component of this name is potentially misleading. First, the object supports any text based format, including XML. Second, it can be used to make requests over both HTTP and HTTPS (some implementations support protocols in addition to HTTP and HTTPS, but that functionality is not covered by this specification). Finally, it supports "requests" in a broad sense of the term as it pertains to HTTP; namely all activity involved with HTTP requests or responses for the defined HTTP methods.

Some simple code to do something with data from an XML document fetched over the network:

```
function processData(data) {
    // taking care of data
}

function handler() {
    if(this.readyState == this.DONE) {
        if(this.status == 200 &&
           this.responseXML != null &&
           this.responseXML.getElementById('test').textContent) {
            // success!
            processData(this.responseXML.getElementById('test').textContent);
            return;
        }
        // something went wrong
        processData(null);
    }
}

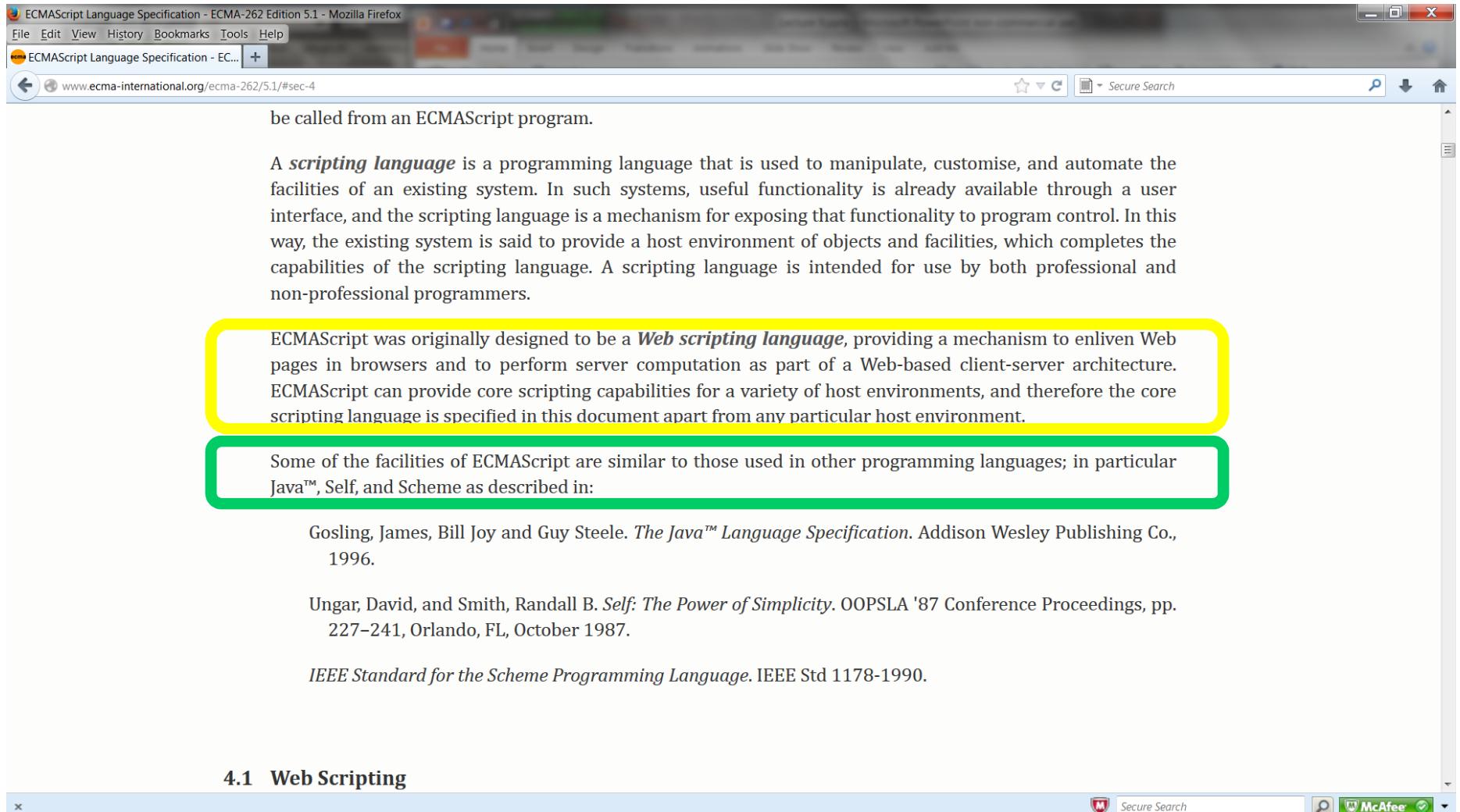
var client = new XMLHttpRequest();
client.onreadystatechange = handler;
client.open("GET", "unicorn.xml");
client.send();
```

If you just want to log a message to the server:

# Origin of XMLHttpRequest?

The screenshot shows a Mozilla Firefox window with the title bar "ECMAScript Language Specification - ECMA-262 Edition 5.1 - Mozilla Firefox". The address bar contains the URL "www.ecma-international.org/ecma-262/5.1/#sec-1". The main content area displays the ECMA International logo and the text "Standard ECMA-262 5.1 Edition / June 2011" followed by "ECMAScript® Language Specification". A large orange callout box in the center states: "This is the HTML rendering of *Ecma-262 Edition 5.1, The ECMAScript Language Specification*. The PDF rendering of this document is located at <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>. The PDF version is the definitive specification. Any discrepancies between this HTML version and the PDF version are unintentional." At the bottom, there is a "Copyright notice" section and a "Secure Search" bar with a McAfee logo.

# Origin of XMLHttpRequest?



The screenshot shows a Mozilla Firefox window displaying the ECMAScript Language Specification - ECMA-262 Edition 5.1. The URL in the address bar is [www.ecma-international.org/ecma-262/5.1/#sec-4](http://www.ecma-international.org/ecma-262/5.1/#sec-4). The page content discusses the definition of a scripting language and its history.

be called from an ECMAScript program.

A *scripting language* is a programming language that is used to manipulate, customise, and automate the facilities of an existing system. In such systems, useful functionality is already available through a user interface, and the scripting language is a mechanism for exposing that functionality to program control. In this way, the existing system is said to provide a host environment of objects and facilities, which completes the capabilities of the scripting language. A scripting language is intended for use by both professional and non-professional programmers.

ECMAScript was originally designed to be a *Web scripting language*, providing a mechanism to enliven Web pages in browsers and to perform server computation as part of a Web-based client-server architecture. ECMAScript can provide core scripting capabilities for a variety of host environments, and therefore the core scripting language is specified in this document apart from any particular host environment.

Some of the facilities of ECMAScript are similar to those used in other programming languages; in particular Java™, Self, and Scheme as described in:

Gosling, James, Bill Joy and Guy Steele. *The Java™ Language Specification*. Addison Wesley Publishing Co., 1996.

Ungar, David, and Smith, Randall B. *Self: The Power of Simplicity*. OOPSLA '87 Conference Proceedings, pp. 227–241, Orlando, FL, October 1987.

*IEEE Standard for the Scheme Programming Language*. IEEE Std 1178-1990.

## 4.1 Web Scripting

# What is XMLHttpRequest?

The screenshot shows a Mozilla Firefox browser window with the title bar "XMLHttpRequest - Mozilla Firefox". The address bar contains the URL "www.w3.org/TR/XMLHttpRequest/". The page content is the W3C XMLHttpRequest Working Draft from December 6, 2012. The page includes the W3C logo, the title "XMLHttpRequest", and the date "W3C Working Draft 6 December 2012". It lists various versions and editors:

- This Version:** <http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/>
- Latest Version:** <http://www.w3.org/TR/XMLHttpRequest/>
- Latest Editor Draft:** <http://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html>
- Previous Versions:**
  - <http://www.w3.org/TR/2012/WD-XMLHttpRequest-20120117/>
  - <http://www.w3.org/TR/2011/WD-XMLHttpRequest2-20110816/>
  - <http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/>
  - <http://www.w3.org/TR/2009/WD-XMLHttpRequest2-20090820/>
  - <http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080930/>
  - <http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080225/>
  - <http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026/>
  - <http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070618/>
  - <http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070227/>
  - <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/>
  - <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060619/>
  - <http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>
- Editor:**
  - [Julian Aubourg \(Creative Area\) <j@ubourg.net>](#)
  - [송정기 \(Jungkee Song\) \(Samsung Electronics Co., Ltd.\) <jungkee.song@samsung.com>](#)
  - [Hallvard R. M. Steen \(Opera Software ASA\) <hallvard@opera.com>](#)
- Previous Editor:** [John Resig](#)

# What is XMLHttpRequest?

The screenshot shows a Mozilla Firefox browser window displaying the W3C XMLHttpRequest Working Draft page. The title bar reads "XMLHttpRequest - Mozilla Firefox". The address bar shows the URL "www.w3.org/TR/XMLHttpRequest/". The left sidebar is labeled "W3C Working Draft". The main content area contains a list of URLs for previous versions of the specification, followed by editor information, previous editor information, and copyright details. Below this is a section titled "Abstract" which defines the XMLHttpRequest specification as an API for transferring data between a client and a server. A yellow box highlights this definition. The "Status of this Document" section provides information on how to comment on the document and links to the WHATWG specification. The bottom of the page includes a "Secure Search" bar and a McAfee security badge.

Editor:  
Julian Aubourg (Creative Area) <j@ubourg.net>  
송정기 (Samsung Electronics Co., Ltd.) <jungkee.song@samsung.com>  
Halvord R. M. Steen (Opera Software ASA) <halvord@opera.com>

Previous Editor:  
Anne van Kesteren (Opera Software ASA) <annevk@annevk.nl>

Copyright © 2012 W3C® (MIT, ERCIM, Keio). All Rights Reserved. W3C liability, trademark and document use rules apply.

**Abstract**

The XMLHttpRequest specification defines an API that provides scripted client functionality for transferring data between a client and a server.

**Status of this Document**

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the [W3C technical reports index](#) at <http://www.w3.org/TR/>.

If you wish to make comments regarding this document in a manner that is tracked by the W3C, please submit them via using [our public bug database](#), or please send comments to [public-webapps@w3.org](mailto:public-webapps@w3.org) ([archived](#)) with [xhr] at the start of the subject line.

The bulk of the text of this specification is also available in the WHATWG [XMLHttpRequest Living Standard](#), under a license that permits reuse of the specification text.

The W3C [Web Applications Working Group](#) is the W3C working group responsible for this specification's progress along the W3C Recommendation track. This specification is the 6 December 2012 Working Draft.

# What is XMLHttpRequest?

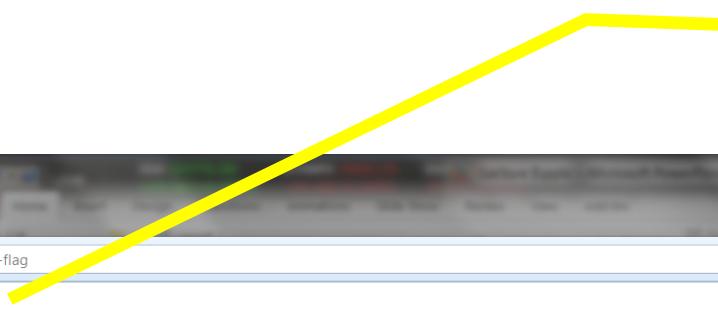
The screenshot shows a Mozilla Firefox browser window displaying the W3C Working Draft for XMLHttpRequest. The URL in the address bar is [www.w3.org/TR/XMLHttpRequest/](http://www.w3.org/TR/XMLHttpRequest/). The left sidebar contains a 'Table of Contents' with several sections, including '1 Introduction', '2 Conformance', '3 Terminology', and '4 Interface XMLHttpRequest'. The '4 Interface XMLHttpRequest' section is expanded, showing sub-sections like '4.1 Task sources', '4.2 Constructors', '4.3 Garbage collector', '4.4 Event handlers', '4.5 States', '4.6 Request', and '4.7 Response'. Three yellow arrows point from the '4.4 Event handlers', '4.6 Request', and '4.7 Response' sections to three separate callout boxes on the right. The first callout box is titled '1. Event Handlers for XMLHttpRequest'. The second is '2. States for XMLHttpRequest'. The third is '3. Methods APIs for XMLHttpRequest'.

Table of Contents

- 1 Introduction
  - 1.1 Specification history
- 2 Conformance
  - 2.1 Dependencies
  - 2.2 Extensibility
- 3 Terminology
- 4 Interface XMLHttpRequest
  - 4.1 Task sources
  - 4.2 Constructors
  - 4.3 Garbage collector
  - 4.4 Event handlers**
  - 4.5 States**
  - 4.6 Request**
    - 4.6.1 The `open()` method
    - 4.6.2 The `setRequestHeader()` method
    - 4.6.3 The `timeout` attribute
    - 4.6.4 The `withCredentials` attribute
    - 4.6.5 The `upload` attribute
    - 4.6.6 The `send()` method
    - 4.6.7 Infrastructure for the `send()` method
    - 4.6.8 The `abort()` method
  - 4.7 Response**
    - 4.7.1 The `status` attribute
    - 4.7.2 The `statusText` attribute
    - 4.7.3 The `getResponseHeader()` method
    - 4.7.4 The `getAllResponseHeaders()` method
    - 4.7.5 Response entity body
    - 4.7.6 The `overrideMimeType()` method
    - 4.7.7 The `responseType` attribute
    - 4.7.8 The `response` attribute
    - 4.7.9 The `responseText` attribute
    - 4.7.10 The `responseXML` attribute
  - 4.8 Events summary
- 5 Interface FormData
- 6 `data:` URLs and HTTP
- References
- Acknowledgments

# What is XMLHttpRequest?

## 1. Open method



The screenshot shows a Mozilla Firefox browser window displaying the W3C XMLHttpRequest specification. The page title is "XMLHttpRequest - Mozilla Firefox". The URL in the address bar is "www.w3.org/TR/XMLHttpRequest/#synchronous-flag". On the left, a vertical bar says "W3C Working Draft". The main content area has a yellow header "4.6.1 The `open()` method". Below it is a red-bordered box containing non-normative implementation requirements. The code for the `open` method is shown in green:

```
client . open(method, url [, async = true [, user = null [, password = null]]])
```

Sets the [request method](#), [request URL](#), [synchronous flag](#), [request username](#), and [request password](#).

Throws a "[SyntaxError](#)" exception if one of the following is true:

- *method* is not a valid HTTP method.
- *url* cannot be resolved.
- *url* contains the "[user:password](#)" format in the [userinfo](#) production.

Throws a "[SecurityError](#)" exception if *method* is a case-insensitive match for CONNECT, TRACE or TRACK.

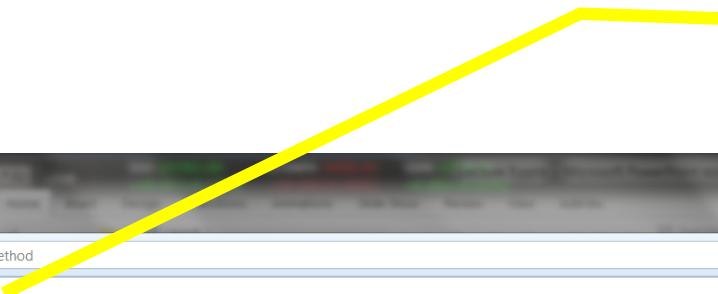
Throws an "[InvalidAccessError](#)" exception if *async* is false, the [JavaScript global environment](#) is a [document environment](#), and either the [anonymous flag](#) is set, the [timeout](#) attribute is not zero, the [withCredentials](#) attribute is true, or the [responseType](#) attribute is not the empty string.

The `open(method, url, async, user, password)` method must run these steps (unless otherwise indicated):

1. Let *base* be null.
2. If the [JavaScript global environment](#) is a [document environment](#), run these steps:
  1. If [document](#) is not [fully active](#), [throw](#) an "[InvalidStateError](#)" exception and terminate the overall set of steps.
  2. Set *base* to the [document base URL](#) of [document](#).
  3. Set [source origin](#) to a globally unique identifier, if the [anonymous flag](#) is set, and the [origin](#) of [document](#) otherwise.
  4. Set [referrer source](#) to [document](#).

# What is XMLHttpRequest?

## 2. Send method



The screenshot shows a Mozilla Firefox browser window displaying the W3C Working Draft for XMLHttpRequest. The page is titled "4.6.6 The `send()` method". A yellow arrow points from the title "2. Send method" to this section. The content includes a code snippet and a note about implementation requirements.

`client . send([data = null])`

This box is non-normative. Implementation requirements are given below this box.

Initiates the request. The optional argument provides the `request entity body`. The argument is ignored if `request method` is `GET` or `HEAD`.  
Throws an "`InvalidStateError`" exception if the state is not `OPENED` or if the `send()` flag is set.

The `send(data)` method must run these steps (unless otherwise noted). This algorithm can be **terminated** by invoking the `open()` or `abort()` method. When it is **terminated** the user agent must terminate the algorithm after finishing the step it is on.

**Note:** The `send()` algorithm can only be terminated if the `synchronous` flag is unset and only after the method call has returned.

1. If the state is not `OPENED`, throw an "`InvalidStateError`" exception and terminate these steps.
2. If the `send()` flag is set, throw an "`InvalidStateError`" exception and terminate these steps.
3. If the `request method` is `GET` or `HEAD`, set `data` to null.
4. If `data` is null, do not include a `request entity body` and go to the next step.

Otherwise, let `encoding` be null, `mime type` be null, and then follow these rules, depending on `data`:

↳ `ArrayBufferView`

Let the `request entity body` be the raw data represented by `data`.

↳ `Blob`

If the object's `type` attribute is not the empty string let `mime type` be its value.

# What is XMLHttpRequest?

readyState of  
XMLHttpRequest Object



**W3C Working Draft**

**4.5 States**

`client . readyState`  
Returns the current state.

This box is non-normative. Implementation requirements are given below this box.

The XMLHttpRequest object can be in several states. The readyState attribute must return the current state, which must be one of the following values:

**UNSENT (numeric value 0)**  
The object has been constructed.

**OPENED (numeric value 1)**  
The open() method has been successfully invoked. During this state request headers can be set using setRequestHeader() and the request can be made using the send() method.

**HEADERS\_RECEIVED (numeric value 2)**  
All redirects (if any) have been followed and all HTTP headers of the final response have been received. Several response members of the object are now available.

**LOADING (numeric value 3)**  
The response entity body is being received.

**DONE (numeric value 4)**  
The data transfer has been completed or something went wrong during the transfer (e.g. infinite redirects).

The send() flag indicates that the send() method has been invoked. It is initially unset and is used during the OPENED state.

The error flag indicates some type of network error or request abortion. It is initially unset and is used during the DONE state.

**4.6 Request**

# What is XMLHttpRequest?

XMLHttpRequest - Mozilla Firefox

File Edit View History Bookmarks Tools Help

W3 XMLHttpRequest

www.w3.org/TR/XMLHttpRequest/#states

Secure Search

4.4 Event handlers

The following are the [event handlers](#) (and their corresponding [event handler event types](#)) that must be supported on objects implementing an interface that inherits from [XMLHttpRequestEventTarget](#) as attributes:

event handler	event handler event type
onloadstart	loadstart
onprogress	progress
onabort	abort
onerror	error
onload	load
ontimeout	timeout
onloadend	loadend

onreadystatechange

EventHandler

The following is the [event handler](#) (and its corresponding [event handler event type](#)) that must be supported as attribute solely by the [XMLHttpRequest](#) object:

event handler	event handler event type
onreadystatechange	readystatechange

4.5 States

client . readyState

Returns the current state.

This box is non-normative. Implementation requirements are given below this box.

The XMLHttpRequest object can be in several states. The readyState attribute must return the current state, which must be one of the following values:

onreadystatechange

18

Highlight All Match Case

Secure Search

McAfee

# What is XMLHttpRequest?

responseText

XMLHttpRequest - Mozilla Firefox  
File Edit View History Bookmarks Tools Help  
W3 XMLHttpRequest  
www.w3.org/TR/XMLHttpRequest/#the-responsetext-attribute

4.7.9 The `responseText` attribute

`client . responseText`  
Returns the [text response entity body](#).  
Throws an "[InvalidStateError](#)" exception if `responseType` is not the empty string or "text".

This box is non-normative. Implementation requirements are given below this box.

The `responseText` attribute must return the result of running these steps:

1. If `responseType` is not the empty string or "text", [throw](#) an "[InvalidStateError](#)" exception and terminate these steps.
2. If the state is not [LOADING](#) or [DONE](#), return the empty string and terminate these steps.
3. If the [error flag](#) is set, return the empty string and terminate these steps.
4. Return the [text response entity body](#).

4.7.10 The `responseXML` attribute

`client . responseXML`  
Returns the [document response entity body](#).  
Throws an "[InvalidStateError](#)" exception if `responseType` is not the empty string or "document".

This box is non-normative. Implementation requirements are given below this box.

The `responseXML` attribute must return the result of running these steps:

1. If `responseType` is not the empty string or "document", [throw](#) an "[InvalidStateError](#)" exception and terminate these steps.
2. If the state is not [DONE](#), return null and terminate these steps.
3. If the [error flag](#) is set, return null and terminate these steps.

onreadystatechange

Highlight All Match Case x  
Secure Search  
McAfee ✓

# What is DOM?

XML DOM Introduction - Mozilla Firefox

File Edit View History Bookmarks Tools Help

XML DOM Introduction

www.w3schools.com/dom/dom\_intro.asp

w3schools.com

HOME HTML CSS JAVASCRIPT JQUERY XML ASP.NET PHP SQL MORE... REFERENCES | EXAMPLES | FORUM | ABOUT

**XML DOM Tutorial**

- DOM HOME
- DOM Introduction**
- DOM Nodes
- DOM Node Tree
- DOM Parser
- DOM Load Function
- DOM Methods
- DOM Accessing
- DOM Info
- DOM Node List
- DOM Traversing
- DOM Browsers
- DOM Navigating

**Manipulate Nodes**

- DOM Get Values
- DOM Change Nodes
- DOM Remove Nodes
- DOM Replace Nodes
- DOM Create Nodes
- DOM Add Nodes
- DOM Clone Nodes
- DOM XMLHttpRequest

**XML DOM Reference**

- DOM Node Types
- DOM Node
- DOM NodeList
- DOM NamedNodeMap
- DOM Document
- DOM DocumentImpl
- DOM DocumentType
- DOM ProcessingInstr
- DOM Element

✓ Ad muted. [Undo](#)  
We'll do our best to show you more relevant ads in the future.  
Help us show you better ads by updating your [ads settings](#).

**Google**

## XML DOM Introduction

« Previous Next Chapter »

The XML DOM defines a standard for accessing and manipulating XML.

### What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- XML
- JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](#).

### What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents like XML and HTML:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The DOM is separated into 3 different parts / levels:

- Core DOM - standard model for any structured document
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.

### What is the HTML DOM?

Search w3schools.com:  
Google Custom Search

Select Language | ▾

**WEB HOSTING**  
Best Web Hosting  
UK Reseller Hosting  
Wow! \$5 SSD Hosting

**WEB BUILDING**  
Download XML Editor  
FREE Website BUILDER  
FREE Website Creator

**WEB RESOURCES**  
Web Statistics  
Web Validation

**xfinity**  
SIGN UP FOR THE  
**XFINITY TRIPLE PLAY**  
**\$99**  
per month for  
12 months

Secure Search

McAfee

# What is DOM?

XML DOM Introduction - Mozilla Firefox

File Edit View History Bookmarks Tools Help

XML DOM Introduction

www.w3schools.com/dom/dom\_intro.asp

DOM Browsers

DOM Navigating

**Manipulate Nodes**

DOM Get Values

DOM Change Nodes

DOM Remove Nodes

DOM Replace Nodes

DOM Create Nodes

DOM Add Nodes

DOM Clone Nodes

DOM XMLHttpRequest

**XML DOM Reference**

DOM Node Types

DOM Node

DOM NodeList

DOM NamedNodeMap

DOM Document

DOM DocumentImpl

DOM DocumentType

DOM ProcessingInstr

DOM Element

DOM Attribute

DOM Text

DOM CDATA

DOM Comment

DOM XMLHttpRequest

DOM ParseError Obj

DOM Parser Errors

DOM Summary

**XML DOM Examples**

DOM Examples

DOM Validator

What You Should Already Know

Before you continue you should have a basic understanding of the following:

- HTML
- XML
- JavaScript

If you want to study these subjects first, find the tutorials on our [Home page](#).

## What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard.

The DOM defines a standard for accessing documents like XML and HTML:

*"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*

The DOM is separated into 3 different parts / levels:

- Core DOM - standard model for any structured document
- XML DOM - standard model for XML documents
- HTML DOM - standard model for HTML documents

The DOM defines the **objects and properties** of all document elements, and the **methods** (interface) to access them.

## What is the HTML DOM?

The HTML DOM defines the **objects and properties** of all HTML elements, and the **methods** (interface) to access them.

If you want to study the HTML DOM, find the HTML DOM tutorial on our [Home page](#).

## What is the XML DOM?

The XML DOM is:

- A standard object model for XML
- A standard programming interface for XML
- Platform- and language-independent
- A W3C standard

The XML DOM defines the **objects and properties** of all XML elements, and the **methods** (interface) to access them.

In other words: **The XML DOM is a standard for how to get, change, add, or delete XML elements.**

« Previous

Next Chapter »

Secure Search

FREE Website Creator

WEB RESOURCES

Web Statistics

Web Validation

xfinity

SIGN UP FOR THE  
**XFINITY TRIPLE PLAY**

\$99 per month for 12 months

\$100 VISA® PREPAID CARD

LEARN MORE ▶

REPLAY

SHARE THIS PAGE

McAfee

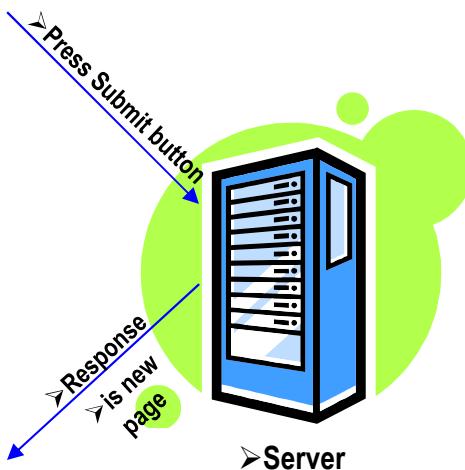
# Traditional Web Apps

## vs. Ajax Apps

- **Traditional Web Apps:  
Infrequent Large Updates**

➤ Web Page 1.

➤ Blah, blah, blah, blah. Yadda, yadda, yadda. Blah, blah, blah, blah. Yadda, yadda, yadda.



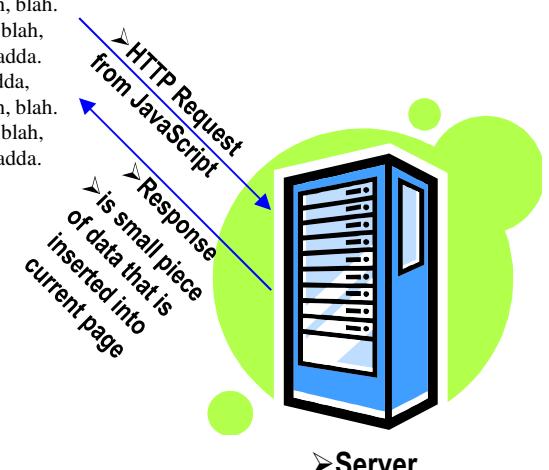
➤ Web Page 2.

➤ Blah, blah, blah, blah. Yadda, yadda, yadda. Blah, blah, blah, blah, blah.

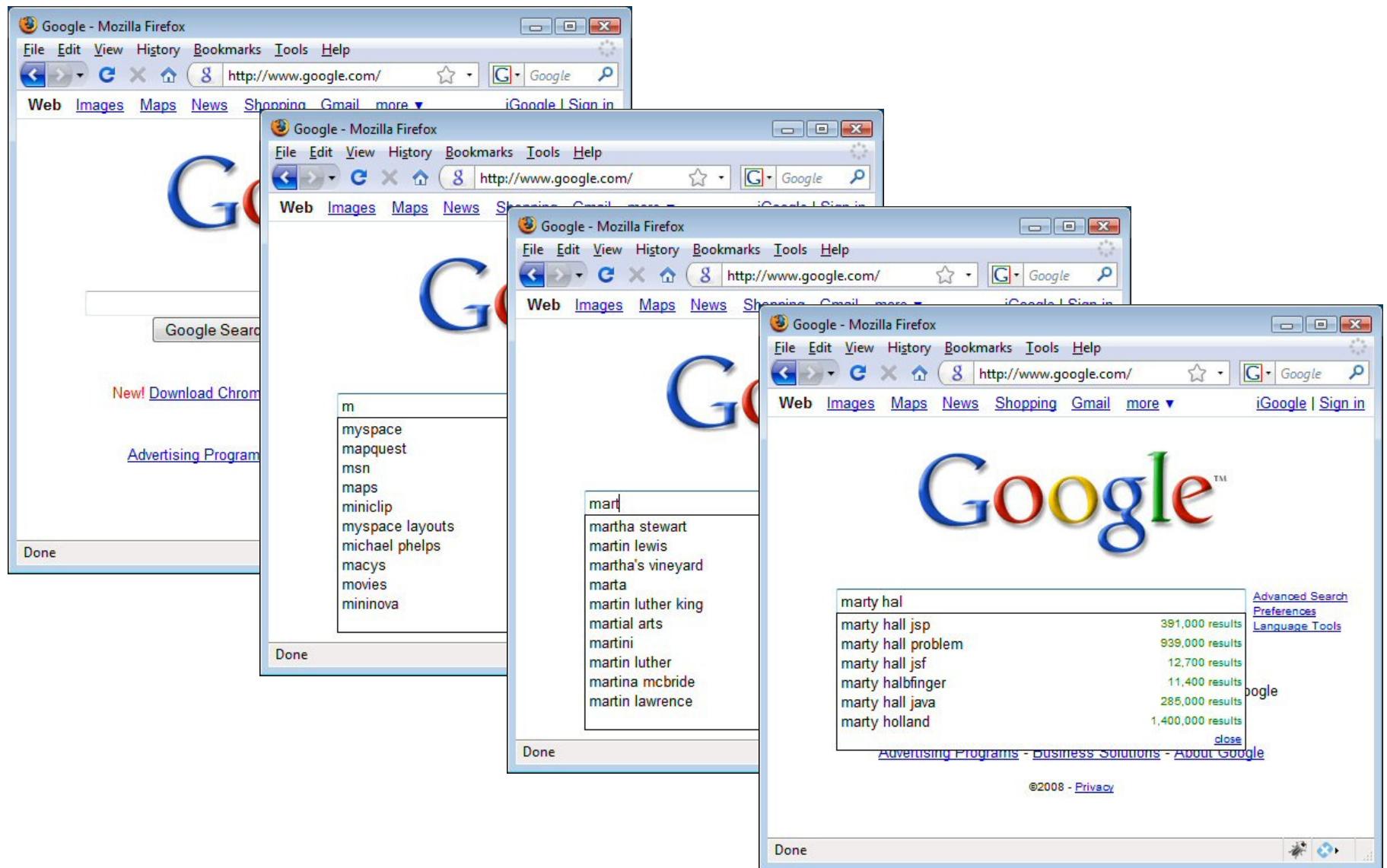
- **Ajax Apps:  
Frequent Small Updates**

➤ Web Page.

➤ Blah, blah, blah, blah. Yadda, yadda, yadda. Blah, blah, blah, blah. Yadda, yadda, yadda.



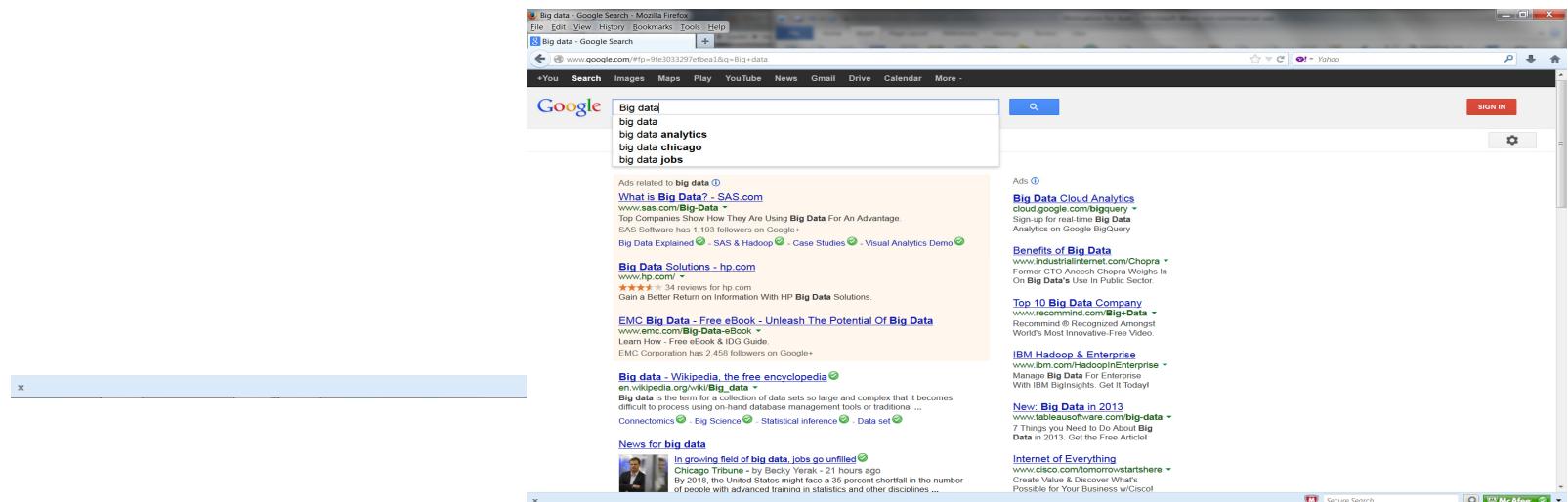
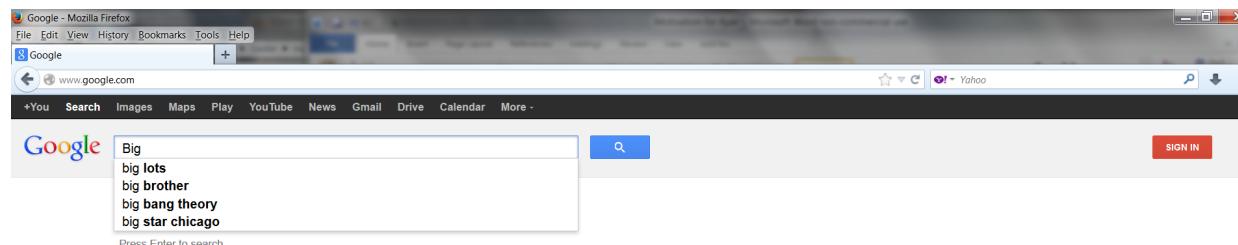
# Google Home Page (formerly Google Suggest)



# Why Ajax?

## • Scenario 1

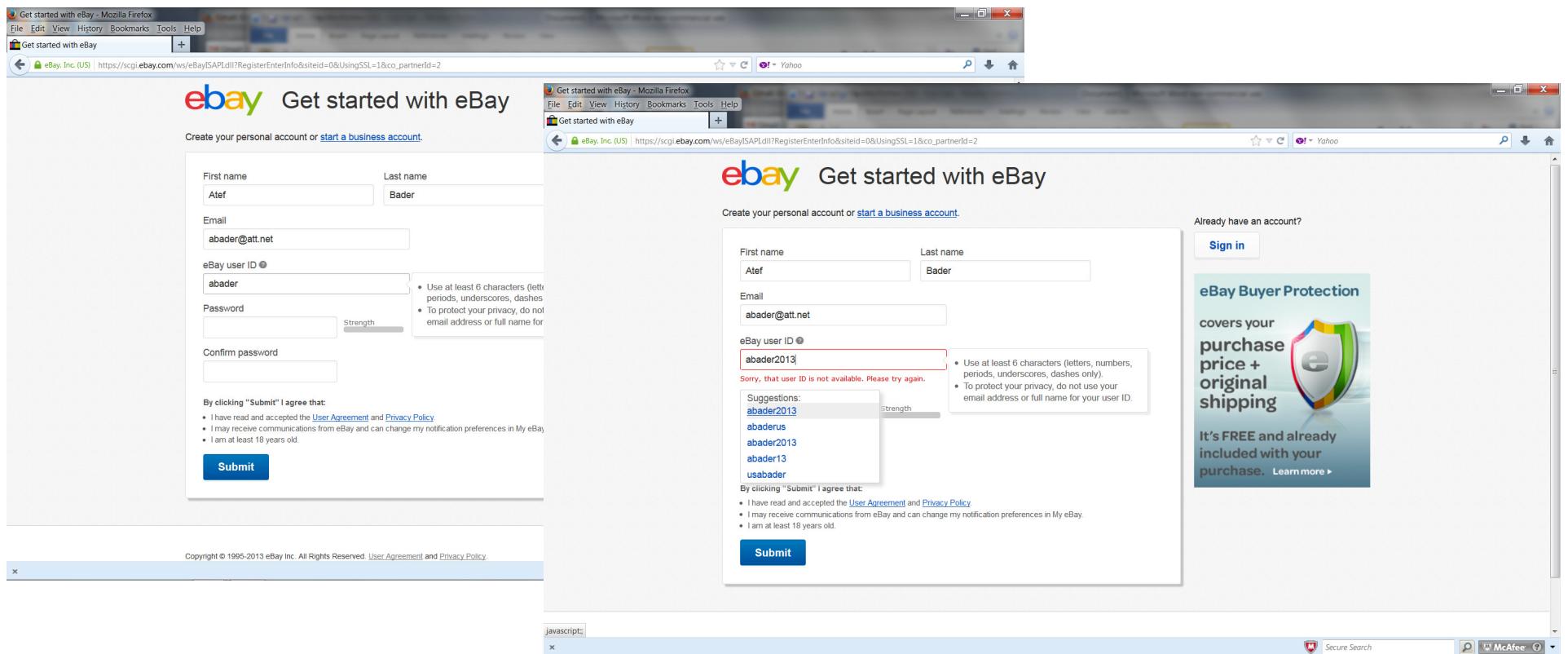
- You did NOT press the enter key yet.; all what you are doing is only typing your search keyword, and AJAX doing the work in the background (asynchronously AJAX/Google trying to guess the next keyword you are trying to type and giving you suggestions and partial results)



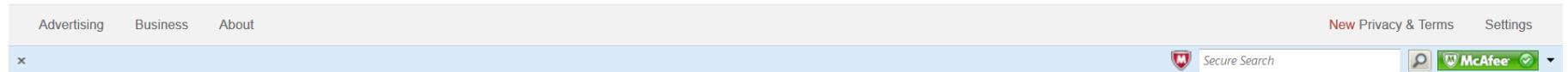
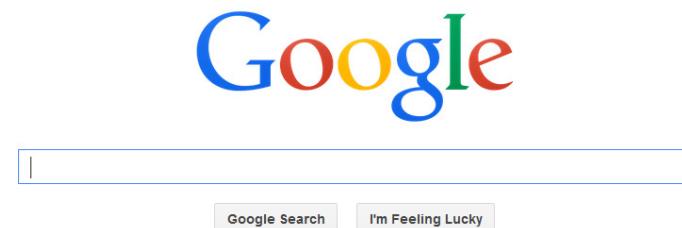
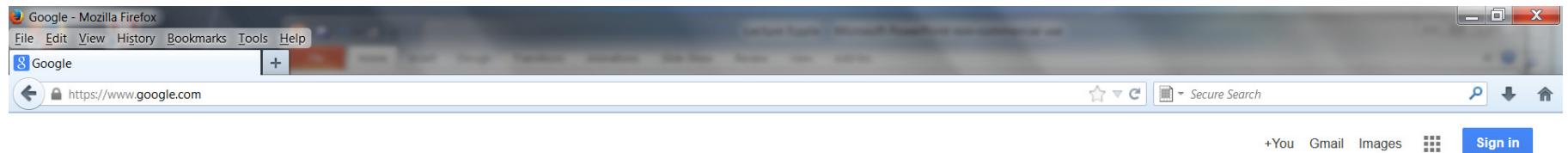
# Why Ajax?

- **Scenario 2**

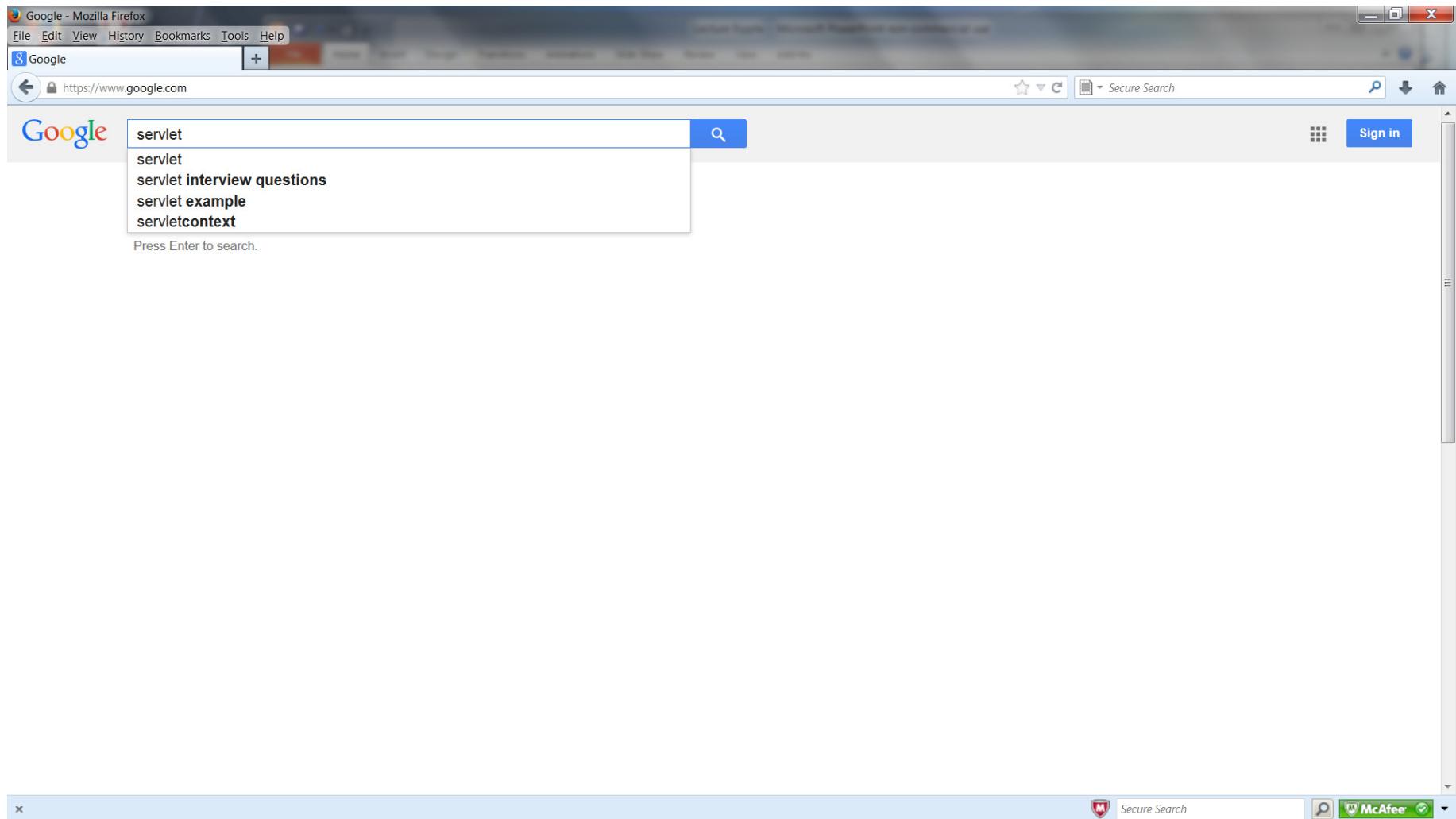
- You are trying to create a new login, and entered the Login ID you want but before going to the next field to enter the password you want, AJAX in the background asynchronously checks with the server to see if the requested Login ID is used by someone else and provides you with a list of potential/candidate Login IDs that you might choose from.



# Google Home Page



# Google Home Page



# Google Home Page

A screenshot of a Mozilla Firefox browser window. The title bar says "servlets - Google Search - Mozilla Firefox". The address bar shows "servlets - Google Search". The search query "servlets" is typed into the search bar. Below the search bar, a dropdown menu lists suggestions: "servlets", "servlets tutorial", "servlets interview questions", and "servlets in java". The main content area displays search results for "Java Servlet".

**Java Servlet - Wikipedia, the free encyclopedia** [\(link\)](#)  
en.wikipedia.org/wiki/Java\_Servlet [\(link\)](#)  
The **Servlet** is a Java programming language class used to extend the capabilities of a server. Although **Servlets** can respond to any types of requests, they are ...  
Introduction [\(link\)](#) - History [\(link\)](#) - Compared with other web ... [\(link\)](#) - Life cycle of a servlet [\(link\)](#)

**Servlets.com** [\(link\)](#)  
www.servlets.com/ [\(link\)](#)  
Servlets.com provides real-life solutions for **Servlet** and JavaServer Pages (JSP) developers. Features news, useful **Servlet** resources, comprehensive links, and ...

**Java Servlet Technology - Oracle** [\(link\)](#)  
www.oracle.com/technetwork/java/index-jsp-135475.html [\(link\)](#)  
The Java **Servlet** API is a Standard Extension to the Java platform that provides web application developers with a simple consistent mechanism for extending ...

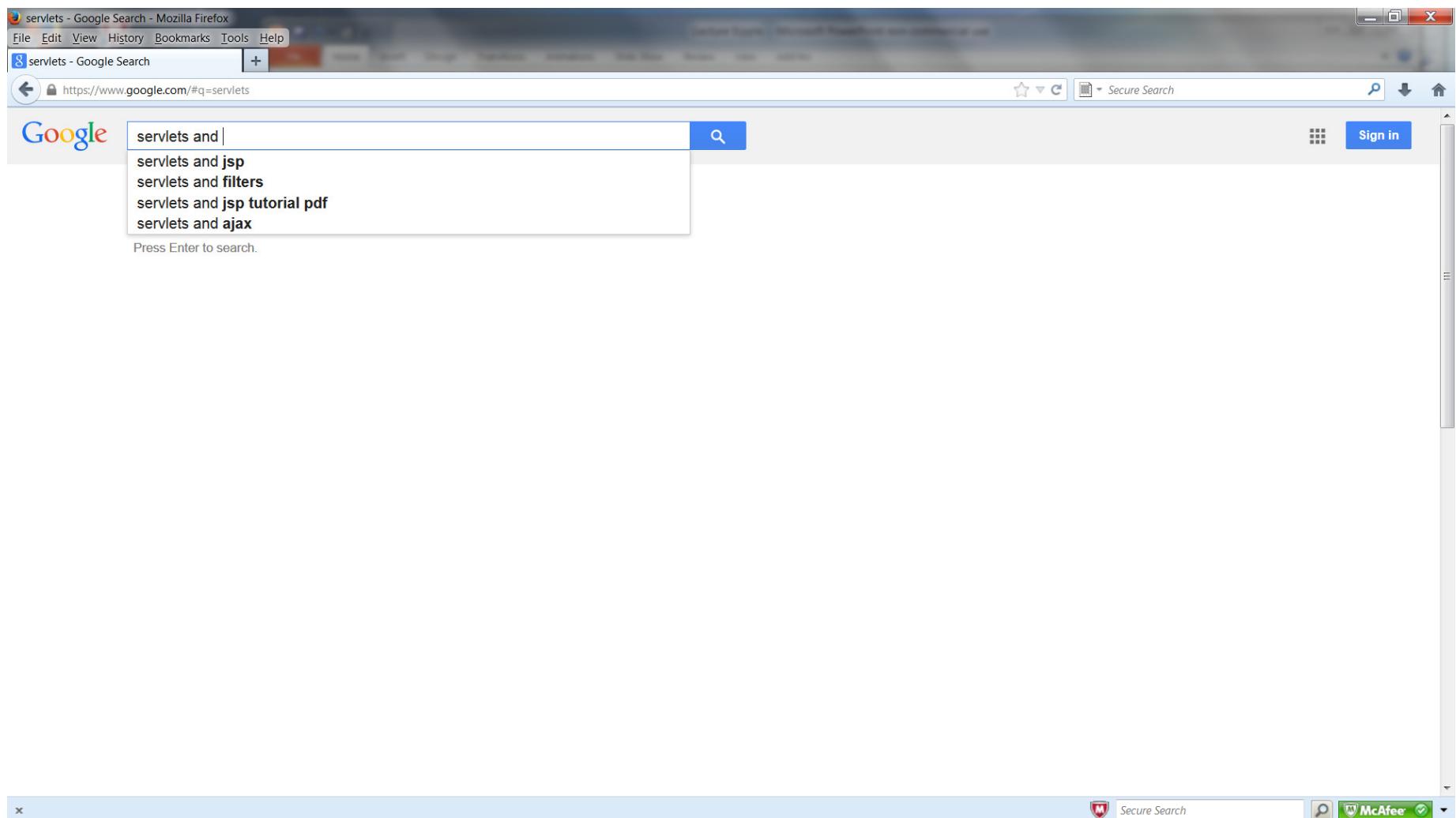
**A Tutorial on Java Servlets and Java Server Pages (JSP) - Apl Jhu** [\(link\)](#)  
www.apl.jhu.edu/~hall/java/Servlet-Tutorial/ [\(link\)](#)  
A detailed tutorial for Java **Servlets** version 2.1/2.2 and JavaServer Pages (JSP) version 1.0.  
Handling Cookies [\(link\)](#) - Session Tracking [\(link\)](#) - JavaServer Pages (JSP) [\(link\)](#) - CGI Variables [\(link\)](#)

**Servlet (Servlet API Documentation)** [\(link\)](#)  
tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Servlet.html [\(link\)](#)  
A **Servlet** is a small Java program that runs within a Web server. **Servlets** receive and respond to requests from Web clients, usually across HTTP, the HyperText ...

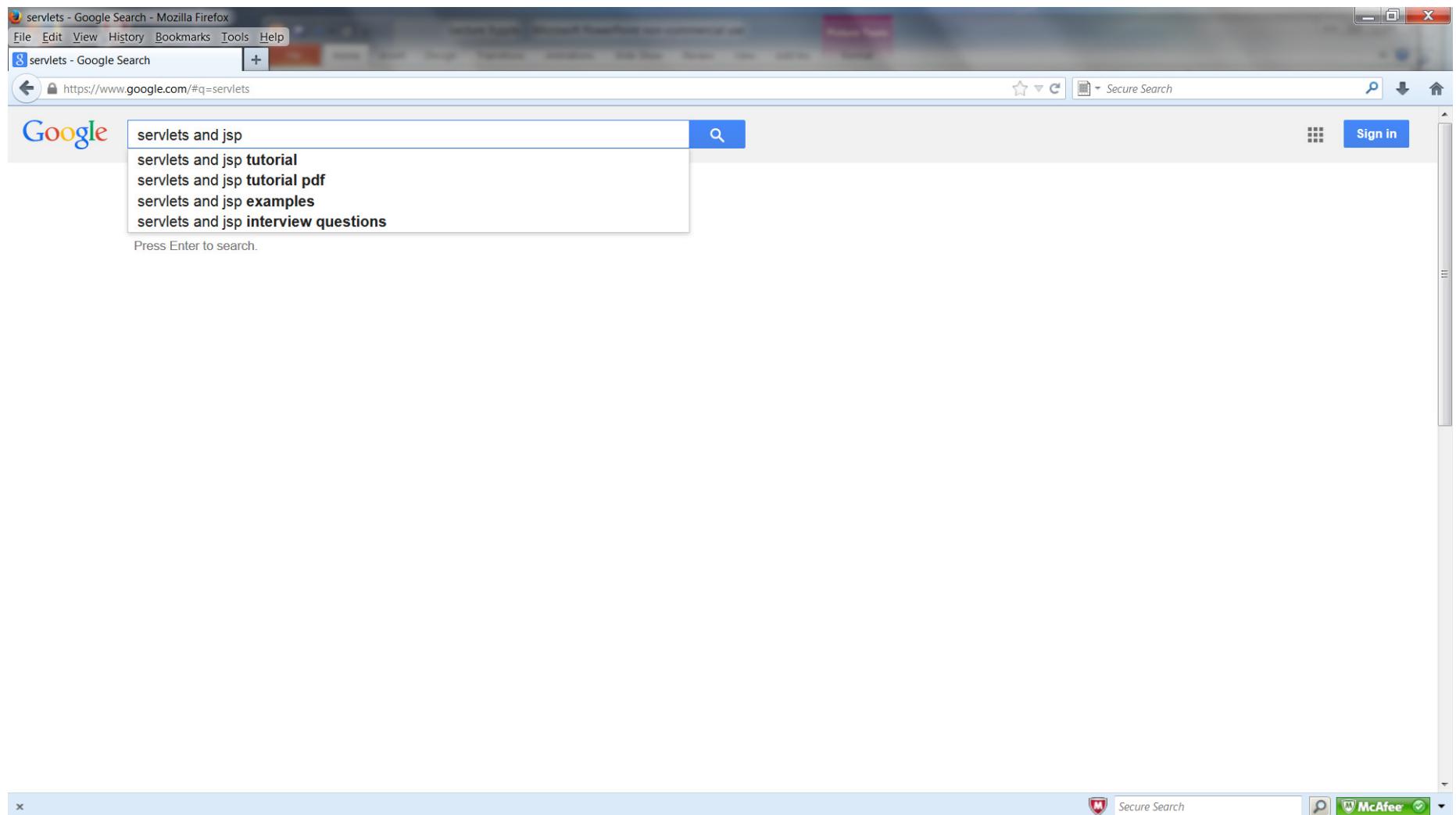
**Servlets Tutorial - Tutorials Point** [\(link\)](#)

The browser status bar at the bottom shows "Secure Search" and "McAfee".

# Google Home Page



# Google Home Page



# The Basic Ajax Process

- **JavaScript**
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
      - Supply as onreadystatechange attribute of request
    - Initiate a GET or POST request
    - Send data
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
    - Do something with result
- **HTML**
  - Load JavaScript
  - Designate control that initiates request
  - Give ids to input elements and to output placeholder region

# The Basic Ajax Process

- **For JavaScripts, 3 Simple Steps:**
  1. getRequestObject
  2. sendRequest
  3. handleResponse

# Define a Request Object

```
function getRequestObject () {  
    if (window.XMLHttpRequest) {  
        return (new XMLHttpRequest());  
    } else if (window.ActiveXObject) {  
        return (new ActiveXObject ("Microsoft.XMLHTTP"));  
    } else {  
        return (null);  
    }  
}
```

➤Fails on older and nonstandard browsers. You don't want to do "throw new Error(...)" here because this is for very old browsers, and Error came only in JavaScript 1.5.

➤Version for Firefox, Netscape 5+, Opera, Safari, Mozilla, Chrome, Internet Explorer 7, and IE 8.

➤Version for Internet Explorer 5.5 and 6

# Initiate Request

```
function sendRequest () {  
    var request = getRequestObject ();  
    request.onreadystatechange =  
        function() { handleResponse(request) };  
    request.open ("GET", "message-data.html", true);  
    request.send(null);  
}
```

➤POST data  
➤(always null for GET requests)

➤URL of server-side resource. Must be on same  
➤server that page was loaded from (\*).

➤Code to call when server responds

➤Don't wait for response  
(Send request asynchronously)

➤(\*) The Same Source (or Same Origin) rule is relaxed somewhat in Firefox 3.5 and later. But, you can only make cross-site requests if the URL you connect to puts in a header specifically allowing it. For details, see <http://www.petefreitag.com/item/703.cfm> and [https://developer.mozilla.org/En/HTTP\\_access\\_control](https://developer.mozilla.org/En/HTTP_access_control)

# Handle Response

```
function handleResponse(request) {  
    if (request.readyState == 4) {  
        alert(request.responseText);  
    }  
}
```

➤Pop up dialog box

➤Text of server response

➤4 means response from server is complete  
➤(handler gets invoked multiple times –  
➤ignore the first ones)

# XMLHttpRequest API and Docs

The screenshot shows a Mozilla Firefox window displaying the XMLHttpRequest Working Draft page from W3C. The title bar reads "XMLHttpRequest - Mozilla Firefox". The address bar shows the URL "http://www.w3.org/TR/XMLHttpRequest/". The left sidebar is blue and labeled "W3C Working Draft". The main content area features the W3C logo at the top, followed by the title "XMLHttpRequest" and the subtitle "W3C Working Draft 6 December 2012". Below this, there are sections for "This Version:", "Latest Version:", "Latest Editor Draft:", and "Previous Versions:", each with a link to the corresponding document. The "Editor:" section lists Julian Aubourg, Jungkee Song, and Hallvard R. M. Steen with their email addresses. The "Previous Editor:" section is listed as empty. The bottom of the browser window includes standard Firefox navigation and search controls.

This Version:  
<http://www.w3.org/TR/2012/WD-XMLHttpRequest-20121206/>

Latest Version:  
<http://www.w3.org/TR/XMLHttpRequest/>

Latest Editor Draft:  
<http://dvcs.w3.org/hg/xhr/raw-file/tip/Overview.html>

Previous Versions:

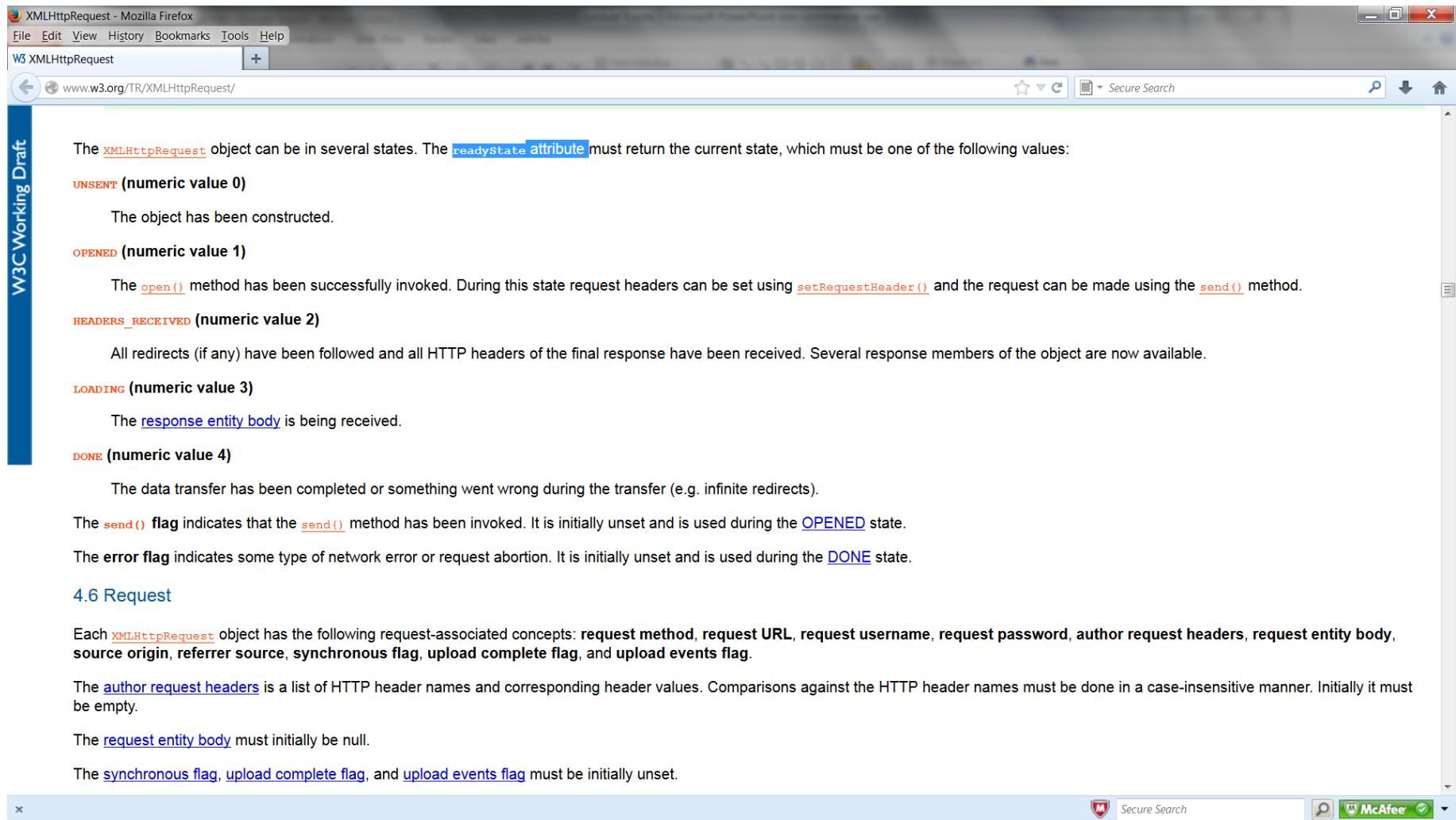
<http://www.w3.org/TR/2012/WD-XMLHttpRequest-20120117/>  
<http://www.w3.org/TR/2011/WD-XMLHttpRequest2-20110816/>  
<http://www.w3.org/TR/2010/WD-XMLHttpRequest2-20100907/>  
<http://www.w3.org/TR/2009/WD-XMLHttpRequest2-20090820/>  
<http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080930/>  
<http://www.w3.org/TR/2008/WD-XMLHttpRequest2-20080225/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20071026/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070618/>  
<http://www.w3.org/TR/2007/WD-XMLHttpRequest-20070227/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060927/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060619/>  
<http://www.w3.org/TR/2006/WD-XMLHttpRequest-20060405/>

Editor:

[Julian Aubourg \(Creative Area\) <j@ubourg.net>](#)  
[송정기 \(Jungkee Song\) \(Samsung Electronics Co., Ltd.\) <jungkee.song@samsung.com>](#)  
[Hallvard R. M. Steen \(Opera Software ASA\) <hallvard@opera.com>](#)

Previous Editor:

# XMLHttpRequest : *readyState* attribute



The screenshot shows a Mozilla Firefox browser window displaying the XMLHttpRequest specification from W3C. The title bar reads "XMLHttpRequest - Mozilla Firefox". The address bar shows "www.w3.org/TR/XMLHttpRequest/". The left sidebar is labeled "W3C Working Draft". The main content area contains the following text:

The `XMLHttpRequest` object can be in several states. The `readystate` attribute must return the current state, which must be one of the following values:

**UNSENT (numeric value 0)**  
The object has been constructed.

**OPENED (numeric value 1)**  
The `open()` method has been successfully invoked. During this state request headers can be set using `setRequestHeader()` and the request can be made using the `send()` method.

**HEADERS\_RECEIVED (numeric value 2)**  
All redirects (if any) have been followed and all HTTP headers of the final response have been received. Several response members of the object are now available.

**LOADING (numeric value 3)**  
The `response entity body` is being received.

**DONE (numeric value 4)**  
The data transfer has been completed or something went wrong during the transfer (e.g. infinite redirects).

The `send()` flag indicates that the `send()` method has been invoked. It is initially unset and is used during the `OPENED` state.

The `error` flag indicates some type of network error or request abortion. It is initially unset and is used during the `DONE` state.

## 4.6 Request

Each `XMLHttpRequest` object has the following request-associated concepts: `request method`, `request URL`, `request username`, `request password`, `author request headers`, `request entity body`, `source origin`, `referrer source`, `synchronous flag`, `upload complete flag`, and `upload events flag`.

The `author request headers` is a list of HTTP header names and corresponding header values. Comparisons against the HTTP header names must be done in a case-insensitive manner. Initially it must be empty.

The `request entity body` must initially be null.

The `synchronous flag`, `upload complete flag`, and `upload events flag` must be initially unset.

# First-class Functions in JavaScript

- JavaScript lets you pass functions around

```
function doSomethingWithResponse() { code }
request.onreadystatechange = doSomethingWithResponse;
```

- This is somewhat similar to function pointers in C/C++
    - Java does not permit this

- JavaScript allows anonymous functions

```
var request = getRequestObject();
request.onreadystatechange =
    function() { code-that-uses-request-variable };
```

- Java has anonymous classes, but no anonymous functions
  - C and C++ have nothing like anonymous functions.
  - Anonymous functions (also called closures) are widely used in Lisp, Ruby, Scheme, C# (as of 2.0), Python, Visual Basic, ML, PHP (as of 5.3), Clojure, Go, & others.

# First-Class Functions: Examples

```
function square(x) { return(x * x); }
function triple(x) { return(x * 3); }
function doOperation(f, x) { return(f(x)); }

doOperation(square, 5); → 25

doOperation(triple, 10); → 30

var functions = [square, triple];

functions[0](10); → 100

functions[1](20); → 60
```

# Anonymous Functions: Examples

```
function square(x) { return(x * x); }

square(10); → 100
(function(x) { return(x * x); })(10); → 100
```

```
function makeMultiplier(n) {
    return(function(x) { return(x * n); });
}
```

```
var factor = 5;
var f = makeMultiplier(factor);
```

```
f(3); → 15
factor = 500;
f(3); → 15
```

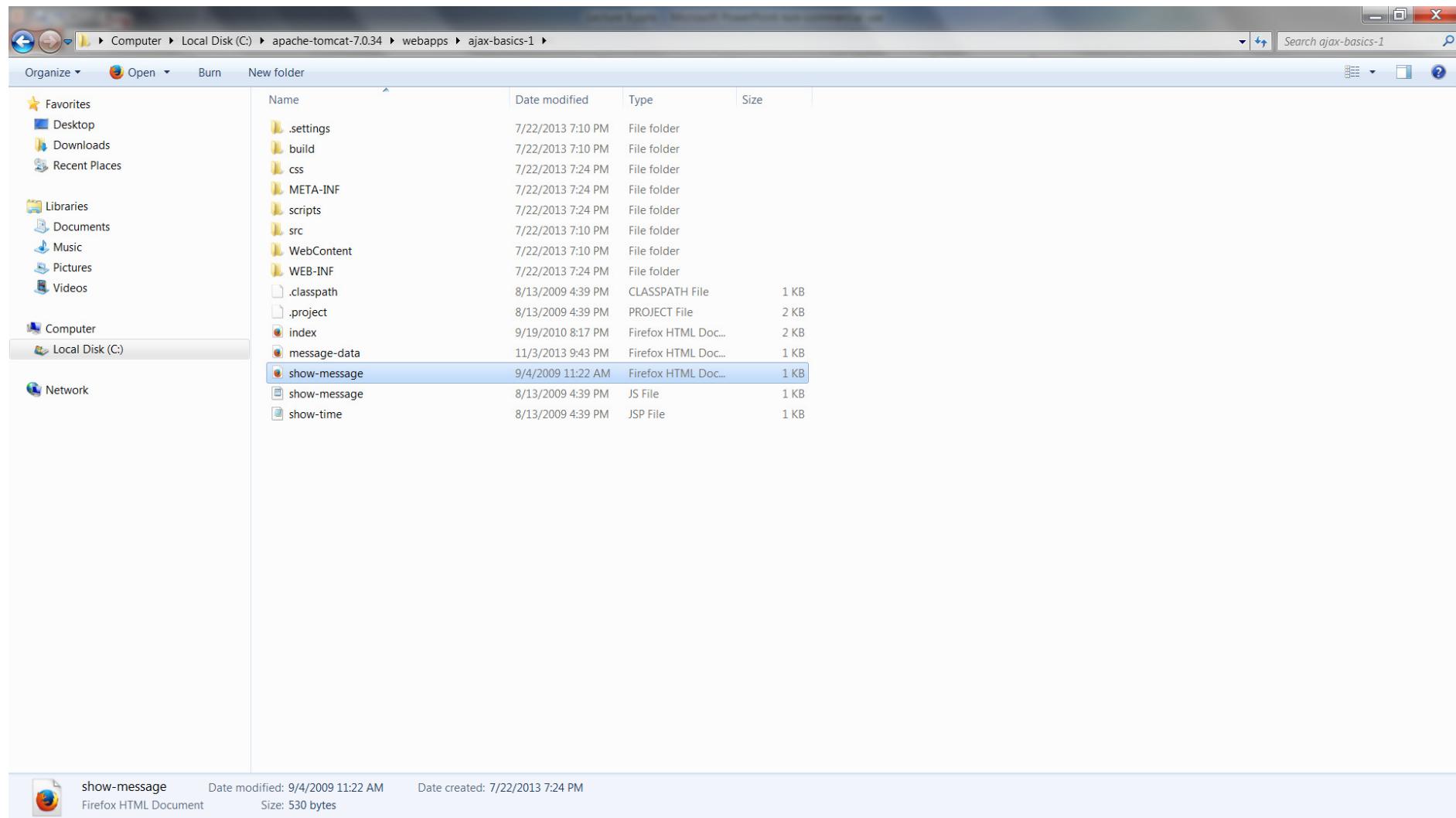
# Local Request Variable

```
function getRequestObject() { ... }

function sendRequest() {
    var request = getRequestObject();
    request.onreadystatechange =
        function() { handleResponse(request); };
    request.open("GET", "...", true);
    request.send(null);
}

function handleResponse(request) {
    ...
}
```

# Our First Simple AJAX WebApp



# Our First Simple AJAX WebApp



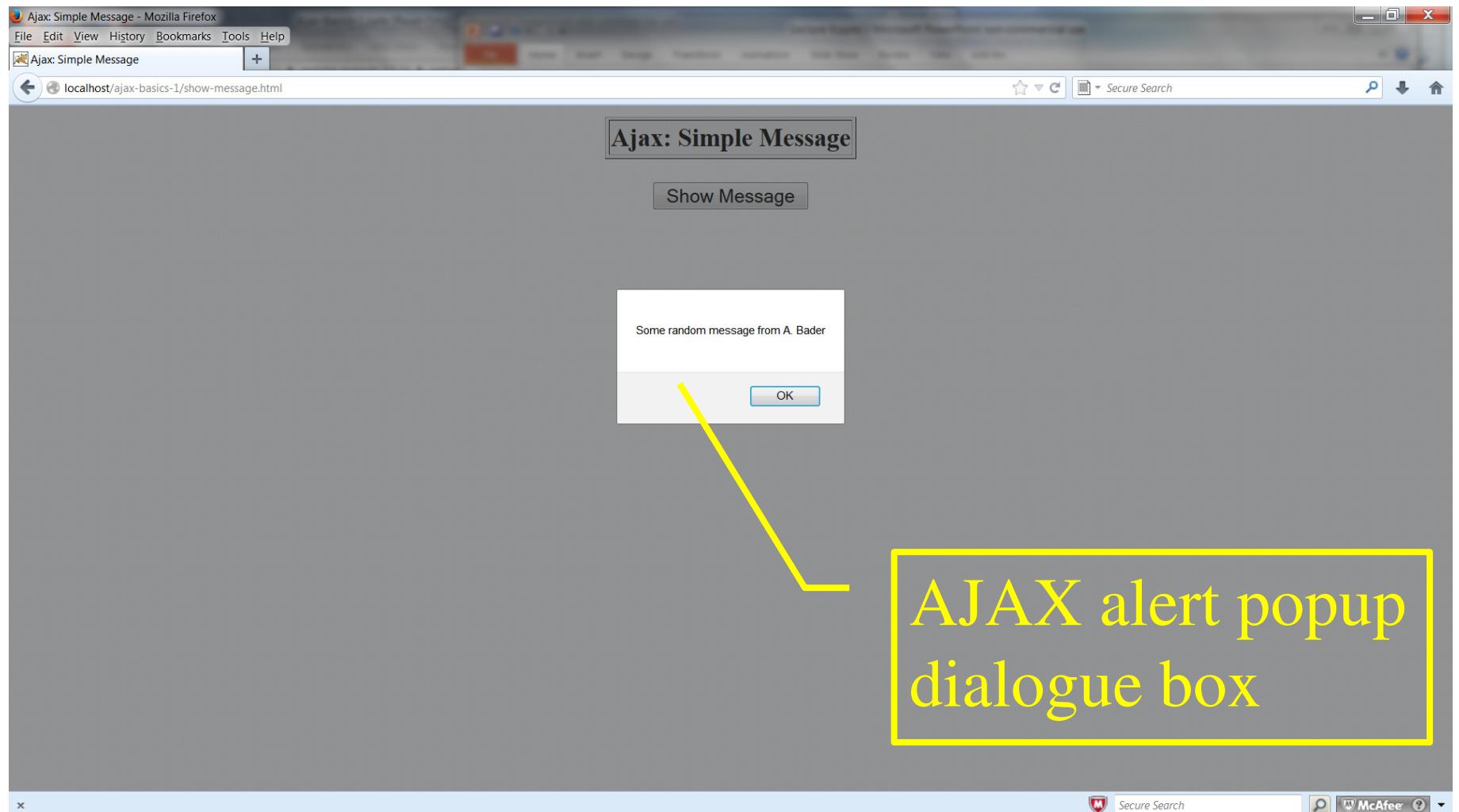
Ajax: Simple Message

Show Message

Click Here



# Our First Simple AJAX WebApp



# Our First Simple AJAX WebApp

- **For this application, 3 files:**
  1. **show-message.html**
  2. **show-message.js**
  3. **message-data.html**

The image shows three separate Notepad windows side-by-side, each containing a different file from the application.

- show-message.html:** This window contains the HTML code for the user interface. It includes a title, a script tag pointing to "show-message.js", and a button labeled "Show Message" with an onclick event handler that calls "sendRequest()".
- show-message.js:** This window contains the JavaScript code for handling the AJAX request. It defines three functions: `getRequestObject`, `sendRequest`, and `handleResponse`. The `getRequestObject` function checks for the presence of ActiveXObject or XMLHttpRequest and returns the appropriate object. The `sendRequest` function creates a new XMLHttpRequest object, sets its.onreadystatechange event to call `handleResponse`, opens a GET request to "message-data.html", and sends the request. The `handleResponse` function checks if the readyState is 4 (complete) and displays an alert with the responseText.
- message-data.html:** This window contains a single line of text: "Some random message from A. Bader".

# Complete JavaScript Code (show-message.js)

```
function getRequestObject() {  
    if (window.XMLHttpRequest) {  
        return(new XMLHttpRequest());  
    } else if (window.ActiveXObject) {  
        return(new ActiveXObject("Microsoft.XMLHTTP"));  
    } else {  
        return(null);  
    }  
}  
  
function sendRequest() {  
    var request = getRequestObject();  
    request.onreadystatechange =  
        function() { handleResponse(request); };  
    request.open("GET", "message-data.html", true);  
    request.send(null);  
}  
  
function handleResponse(request) {  
    if (request.readyState == 4) {  
        alert(request.responseText);  
    }  
}
```

# HTML Code

- Use XHTML, not HTML 4

- In order to manipulate it with DOM

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">...</html>
```

- Due to IE bug, do not use XML header before the DOCTYPE

- Load the JavaScript file

```
<script src="relative-url-of-JavaScript-file"  
       type="text/javascript"></script>
```

- Use separate </script> end tag

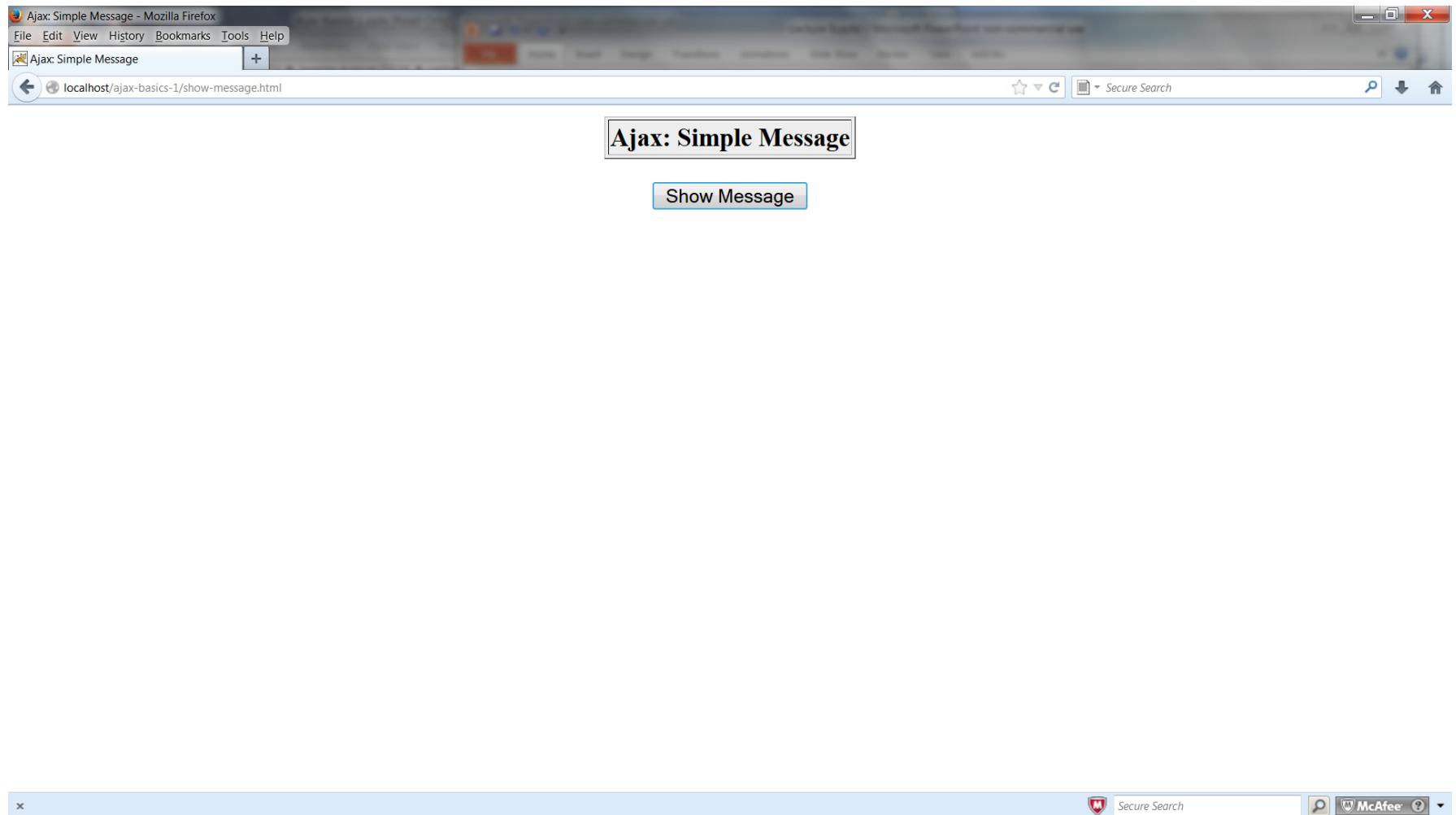
- Designate control to initiate request

```
<input type="button" value="button label"  
      onclick="mainFunction()">
```

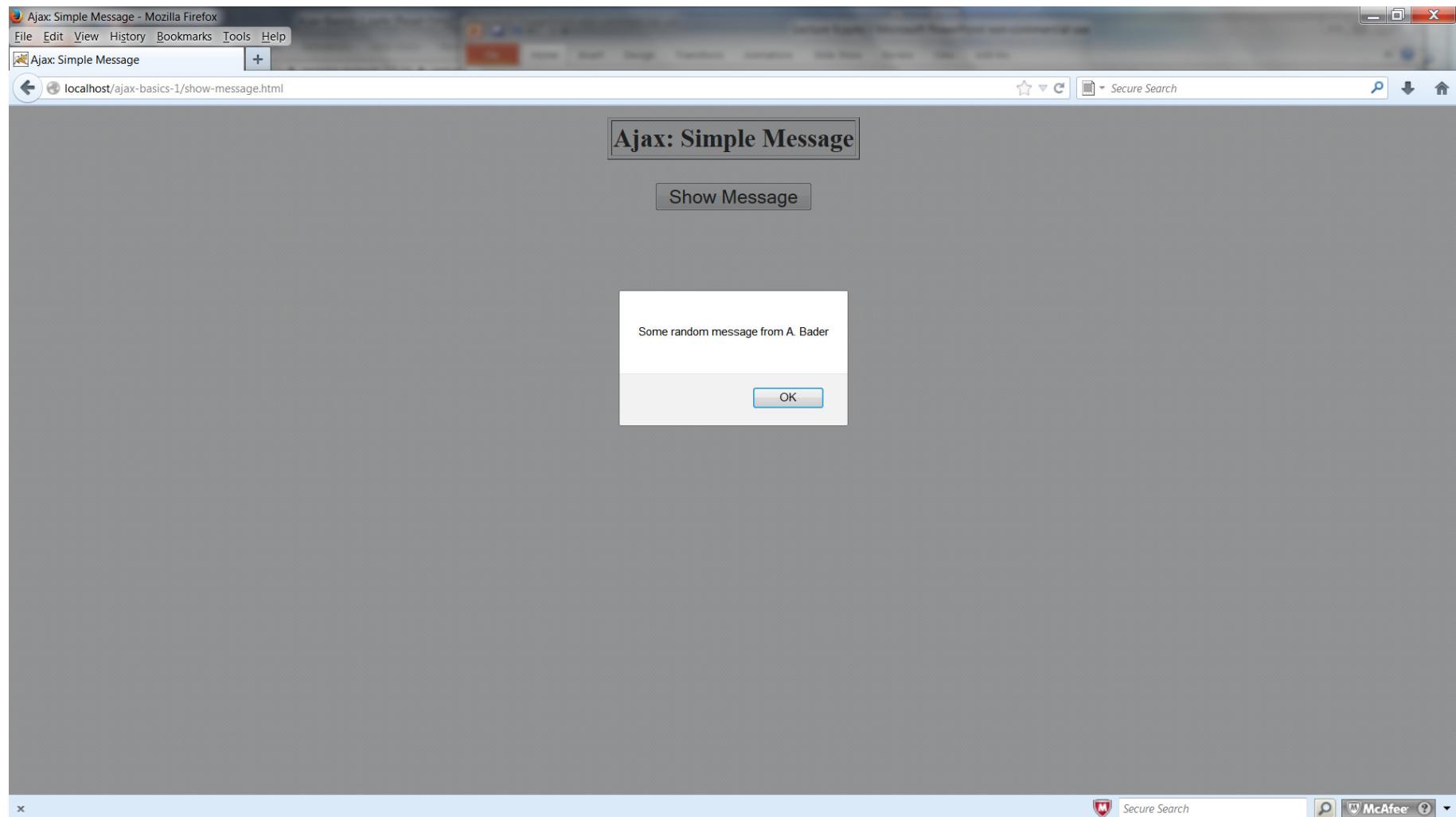
# HTML Code (show-message.html)

```
<!DOCTYPE html PUBLIC "..."  
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
<html xmlns="http://www.w3.org/1999/xhtml">  
<head><title>Ajax: Simple Message</title>  
<script src="show-message.js"  
       type="text/javascript"></script>  
</head>  
<body>  
<center>  
<table border="1" bgcolor="gray">  
  <tr><th><big>Ajax: Simple Message</big></th></tr>  
</table>  
<p/>  
<input type="button" value="Show Message"  
      onclick="sendRequest ()"/>  
</center></body></html>
```

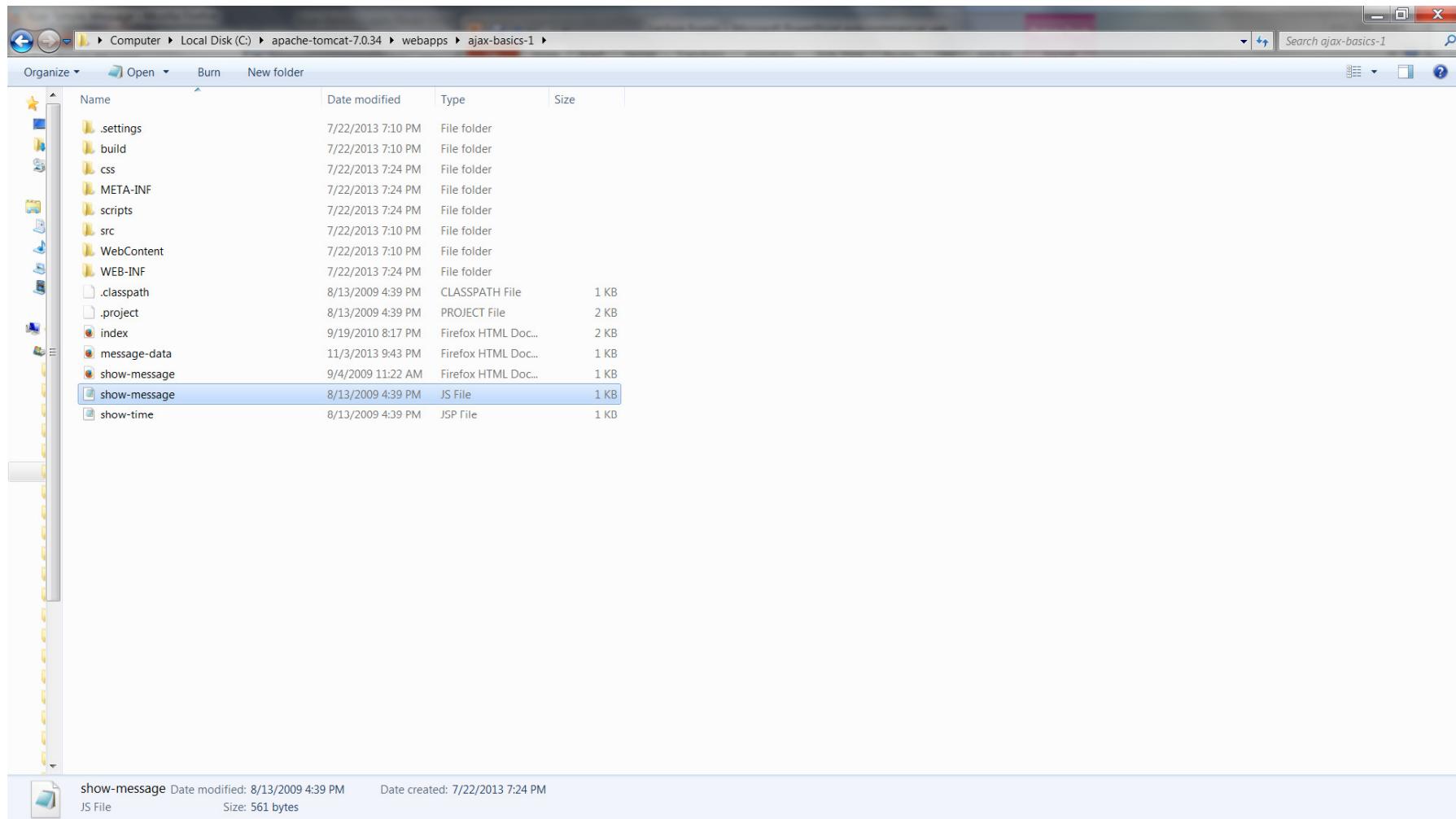
# The Basic Process: Results



# The Basic Process: Results



# WebApp C:\apache-tomcat-7.0.34\webapps\ajax-basics-1



# Dynamic Content from JSP

# First Example: Design Deficiencies

- Content was the same on each request
  - Could have just hardcoded the alert value in JavaScript
  - Instead, invoke a JSP page on the server
- Resource address hardcoded in JavaScript
  - Prevents functions from applying to multiple situations
  - Instead, make generic function and pass address to it
- JavaScript file was in same folder as HTML
  - Makes it hard to reuse the JavaScript in different pages
  - Instead, make a special directory for JavaScript
- No style sheet was used
  - Less for JavaScript to work with when manipulating page
  - Use CSS for normal reasons as well as for JavaScript

# Steps

- JavaScript
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
      - Supply as onreadystatechange attribute of request
    - Initiate a GET or POST request **to a JSP page**
      - **Get the address from a variable instead of hardcoding it**
    - Send data
  - Handle response
    - Wait for readyState of 4 **and HTTP status of 200**
    - Extract return text with responseText or responseXML
    - Do something with result
- HTML
  - Load JavaScript **from centralized directory. Use style sheet.**
  - Designate control that initiates request
  - Give id to output placeholder region

# HTTP/1.1 Status-Code

The screenshot shows a Mozilla Firefox window displaying the RFC 2616 status codes document. The title bar reads "Mozilla Firefox" and the address bar shows "RFC http://www.rfc-editor.org/rfc/rfc2616.txt". The page content starts with "valid request" and then lists the status codes with their descriptions. The status code "200" is highlighted in blue.

```
valid request

The individual values of the numeric status codes defined for
HTTP/1.1, and an example set of corresponding Reason-Phrase's, are
presented below. The reason phrases listed here are only
recommendations -- they MAY be replaced by local equivalents without
affecting the protocol.

Status-Code      =
  "100" ; Section 10.1.1: Continue
  | "101" ; Section 10.1.2: Switching Protocols
  | "200" ; Section 10.2.1: OK
  | "201" ; Section 10.2.2: Created
  | "202" ; Section 10.2.3: Accepted
  | "203" ; Section 10.2.4: Non-Authoritative Information
  | "204" ; Section 10.2.5: No Content
  | "205" ; Section 10.2.6: Reset Content
  | "206" ; Section 10.2.7: Partial Content
  | "300" ; Section 10.3.1: Multiple Choices
  | "301" ; Section 10.3.2: Moved Permanently
  | "302" ; Section 10.3.3: Found
  | "303" ; Section 10.3.4: See Other
  | "304" ; Section 10.3.5: Not Modified
  | "305" ; Section 10.3.6: Use Proxy
  | "307" ; Section 10.3.8: Temporary Redirect
  | "400" ; Section 10.4.1: Bad Request
  | "401" ; Section 10.4.2: Unauthorized
  | "402" ; Section 10.4.3: Payment Required
  | "403" ; Section 10.4.4: Forbidden
  | "404" ; Section 10.4.5: Not Found
```

# Define a Request Object

```
function getRequestObject() {  
    if (window.XMLHttpRequest) {  
        return (new XMLHttpRequest());  
    } else if (window.ActiveXObject) {  
        return (new  
ActiveXObject("Microsoft.XMLHTTP"));  
    } else {  
        return (null);  
    }  
}
```

➤ [No changes from previous example.](#)  
➤ [This code stays the same for entire section.](#)

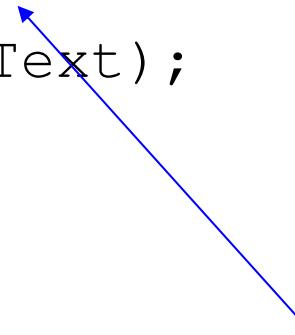
# Initiate Request

```
function ajaxAlert(address) {  
    var request = getRequestObject();  
    request.onreadystatechange =  
        function() { showResponseAlert(request);  
    };  
    request.open("GET", address, true);  
    request.send(null);  
}
```

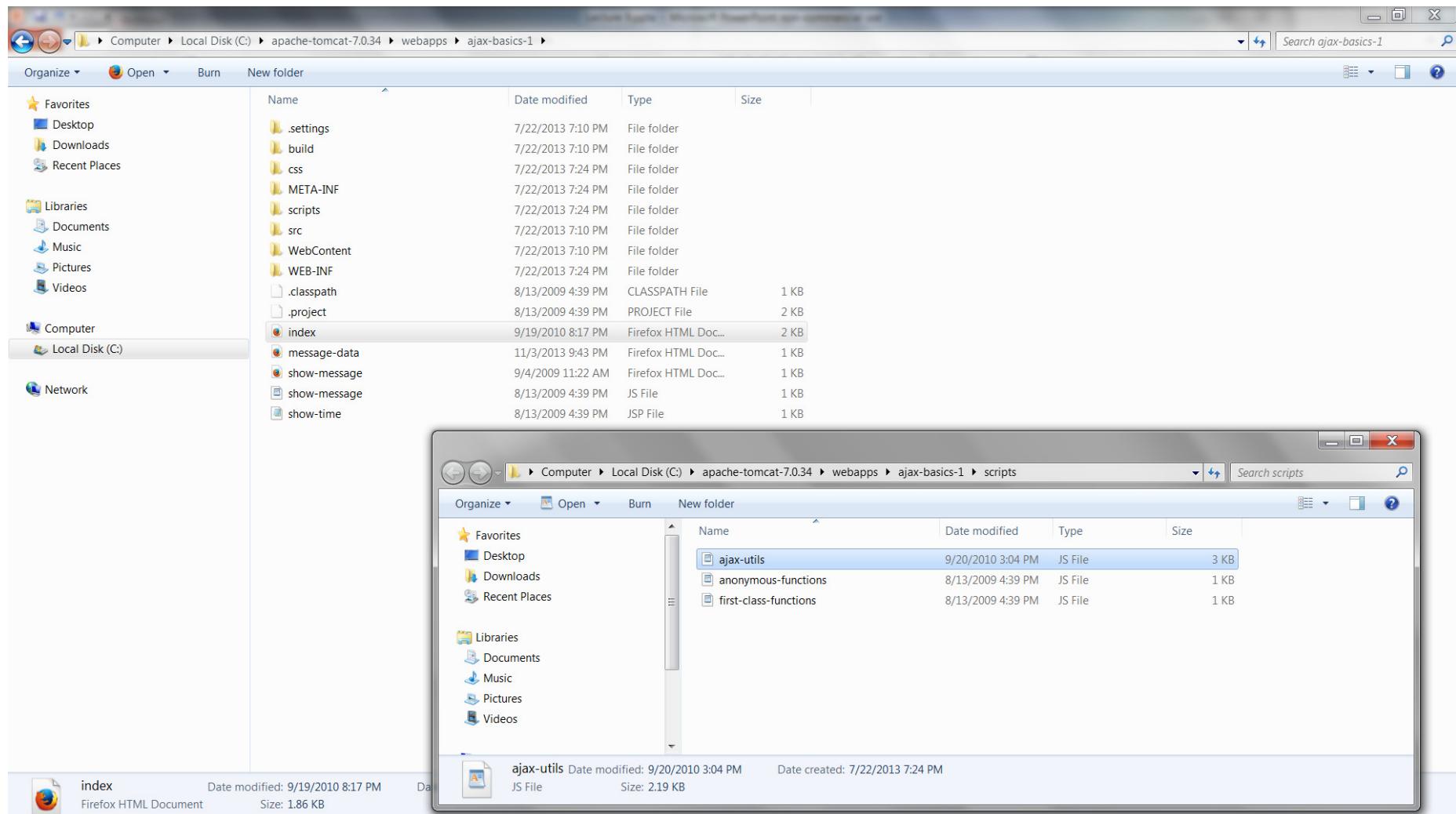
➤Relative URL of server-side resource.  
(In this example, we will pass in the address of a JSP page.)

# Handle Response

```
function showResponseAlert (request) {  
    if ((request.readyState == 4) &&  
        (request.status == 200)) {  
        alert(request.responseText);  
    }  
}
```

- 
- Server response came back with no errors  
(HTTP status code 200).

# Source Code Directories



# Source Code Directories

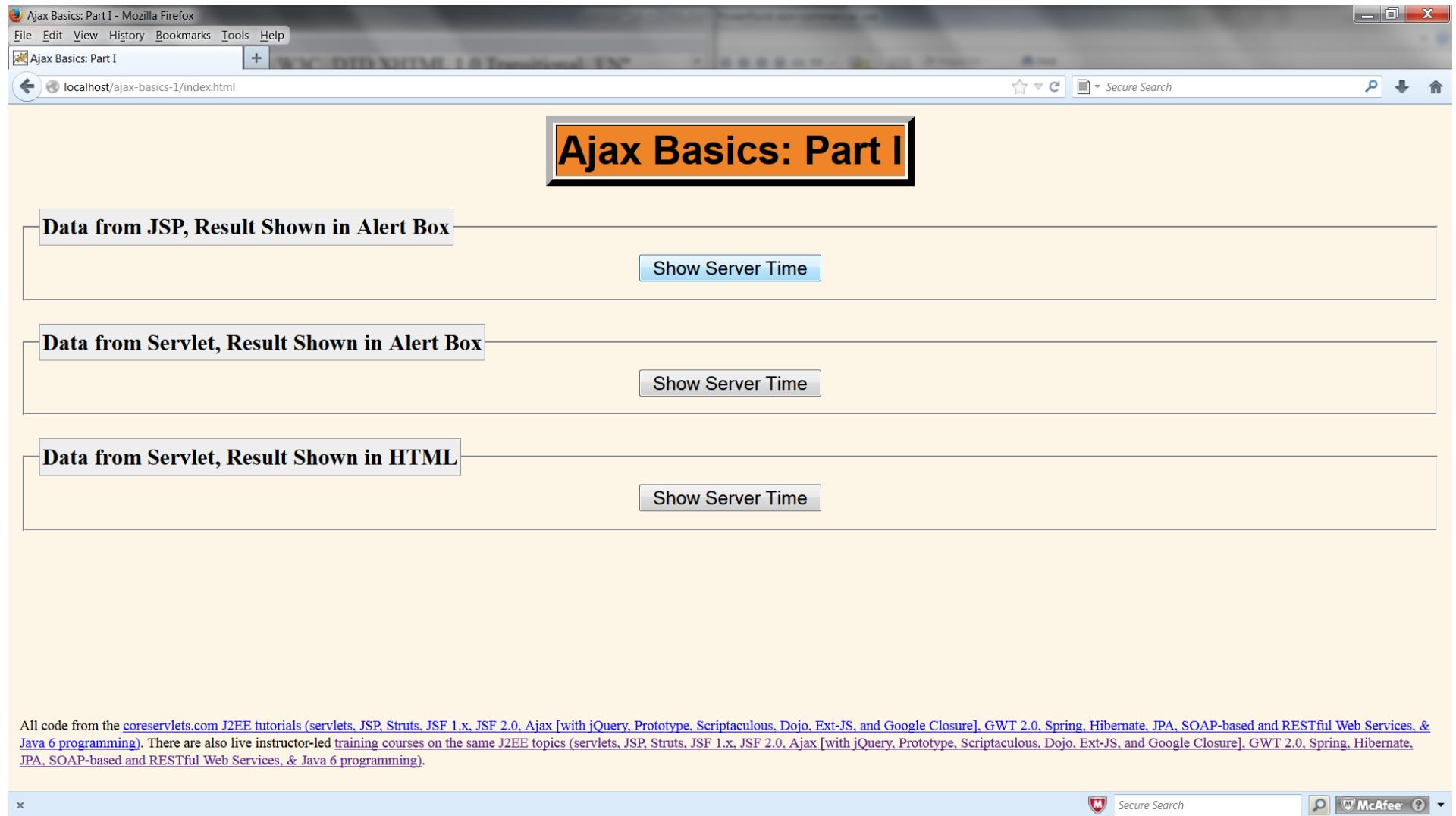
- **For this application, 3 files:**
  1. **index.html**
  2. **ajax-utils.js**
  3. **show-time.jsp**

The image shows three separate Notepad windows side-by-side, each containing a different file's source code.

- index - Notepad**: Contains the HTML code for the main page. It includes a DOCTYPE declaration, an HTML structure with a title, a link to a stylesheet, and a script tag pointing to 'ajax-utils.js'. It also features two form fields: one for displaying data from a JSP and another for displaying data from a Servlet.
- ajax-utils - Notepad**: Contains JavaScript code for handling HTTP requests. It defines a function 'getHttpRequestObject' to return an XMLHttpRequest object, which falls back to ActiveXObject for very old browsers. It also contains an 'ajaxAlert' function that sends an HTTP GET request to a specified address and displays the response in an alert box.
- show-time - Notepad**: Contains JavaServer Page (JSP) code. It uses a scriptlet to create a new Date object and output its value.

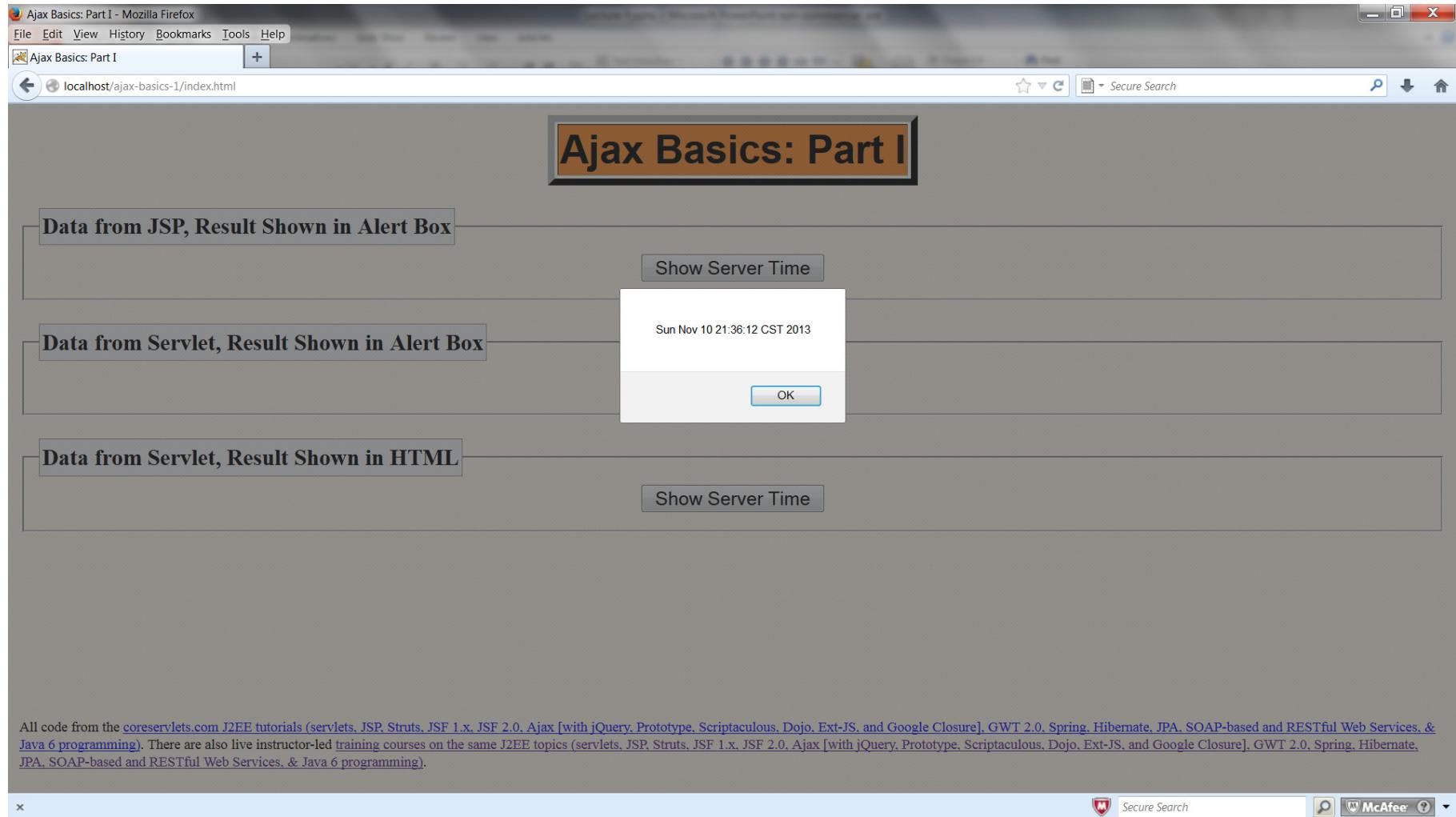
# Source Code Directories

- **Sample Output**



# Source Code Directories

- **Sample Output**



# Complete JavaScript Code

## (Part of ajax-utils.js)

```
function getRequestObject() {
    if (window.XMLHttpRequest) {
        return(new XMLHttpRequest());
    } else if (window.ActiveXObject) {
        return(new ActiveXObject("Microsoft.XMLHTTP"));
    } else {
        return(null);
    }
}

function ajaxAlert(address) {
    var request = getRequestObject();
    request.onreadystatechange =
        function() { showResponseAlert(request); }
    request.open("GET", address, true);
    request.send(null);
}

function showResponseAlert(request) {
    if ((request.readyState == 4) &&
        (request.status == 200)) {
        alert(request.responseText);
    }
}
```

# HTML Code index.html

- Load JavaScript from central location

```
<script src="../scripts/ajax-utils.js"  
       type="text/javascript"></script>
```

- Pass JSP address to main function

```
<input type="button" value="Show Server Time"  
      onclick='ajaxAlert("show-time.jsp")'/>
```

- Use style sheet

```
<link rel="stylesheet"  
      href=".css/styles.css"  
      type="text/css"/>
```

➤ Note single quotes (because of  
double quotes inside parens).

# HTML Code

```
<!DOCTYPE html PUBLIC "..."><html xmlns="http://www.w3.org/1999/xhtml">...<link rel="stylesheet" href=".css/styles.css" type="text/css"/><script src=".scripts/ajax-utils.js" type="text/javascript"></script>...<body>...<fieldset>    <legend>Data from JSP, Result Shown in Alert Box </legend>    <form action="#">        <input type="button" value="Show Server Time" onclick='ajaxAlert("show-time.jsp")' />    </form></fieldset>...</body>
```

# JSP Code (show-time.jsp)

```
<%= new java.util.Date() %>
```

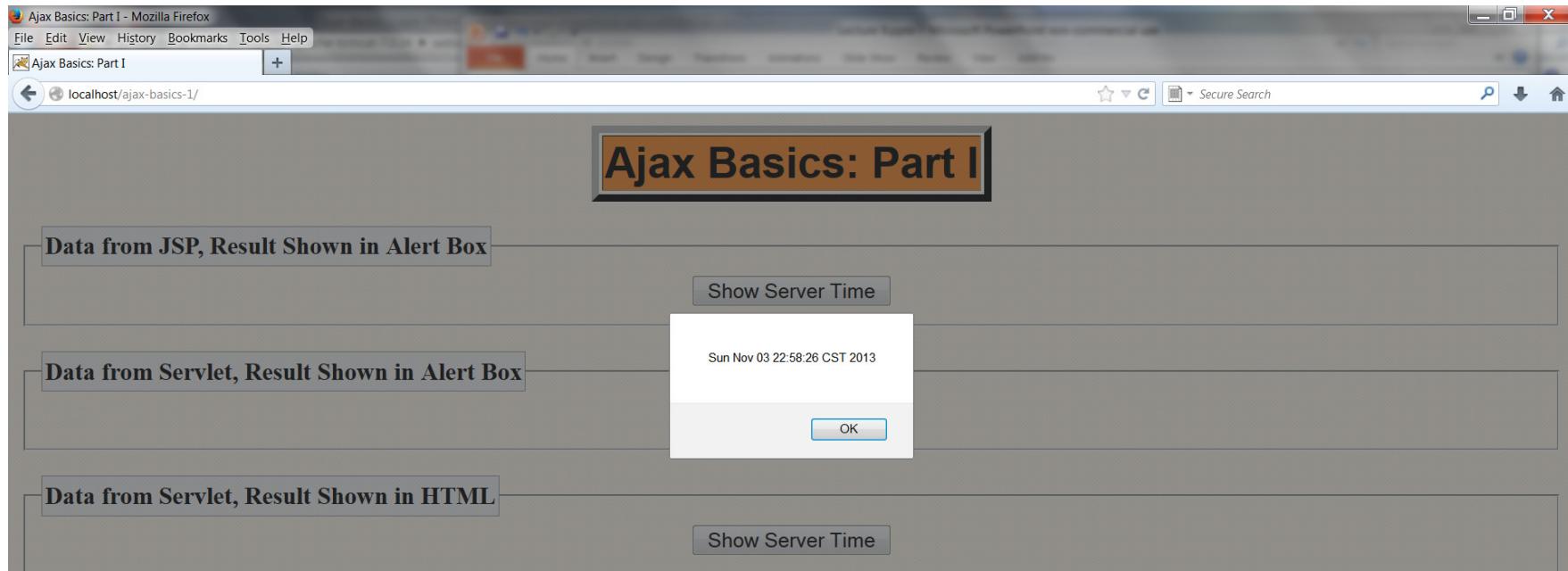
# Message from JSP: Results

A screenshot of a Mozilla Firefox browser window titled "Ajax Basics: Part I - Mozilla Firefox". The address bar shows "localhost/ajax-basics-1/". The page content displays three separate sections, each with a "Show Server Time" button:

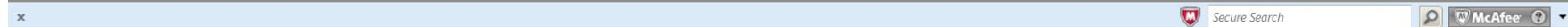
- Data from JSP, Result Shown in Alert Box**: This section contains a button labeled "Show Server Time".
- Data from Servlet, Result Shown in Alert Box**: This section contains a button labeled "Show Server Time".
- Data from Servlet, Result Shown in HTML**: This section contains a button labeled "Show Server Time".

All code from the [coreservlets.com J2EE tutorials](#) (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming). There are also live instructor-led [training courses](#) on the same J2EE topics (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming).

# Message from JSP: Results



All code from the [coreservlets.com J2EE tutorials](#) (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming). There are also live instructor-led training courses on the same J2EE topics (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming).



# Dynamic Content from Servlet

# Steps

- JavaScript
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
      - Supply as onreadystatechange attribute of request
    - Initiate a GET or POST request **to a servlet**
    - Send data
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
    - Do something with result
- HTML
  - Load JavaScript from centralized directory. Use style sheet.
  - Designate control that initiates request
  - Give id to output placeholder region

# Define a Request Object, Initiate Request, Handle Response

```
function getRequestObject() {  
    if (window.XMLHttpRequest) {  
        return(new XMLHttpRequest());  
    } else if (window.ActiveXObject) {  
        return(new ActiveXObject("Microsoft.XMLHTTP"));  
    } else {  
        return(null);  
    }  
}
```

```
function ajaxAlert(address) {  
    var request = getRequestObject();  
    request.onreadystatechange =  
        function() { showResponseAlert(request); }  
    request.open("GET", address, true);  
    request.send(null);  
}
```

```
function showResponseAlert(request) {  
    if ((request.readyState == 4) &&  
        (request.status == 200)) {  
        alert(request.responseText);  
    }  
}
```

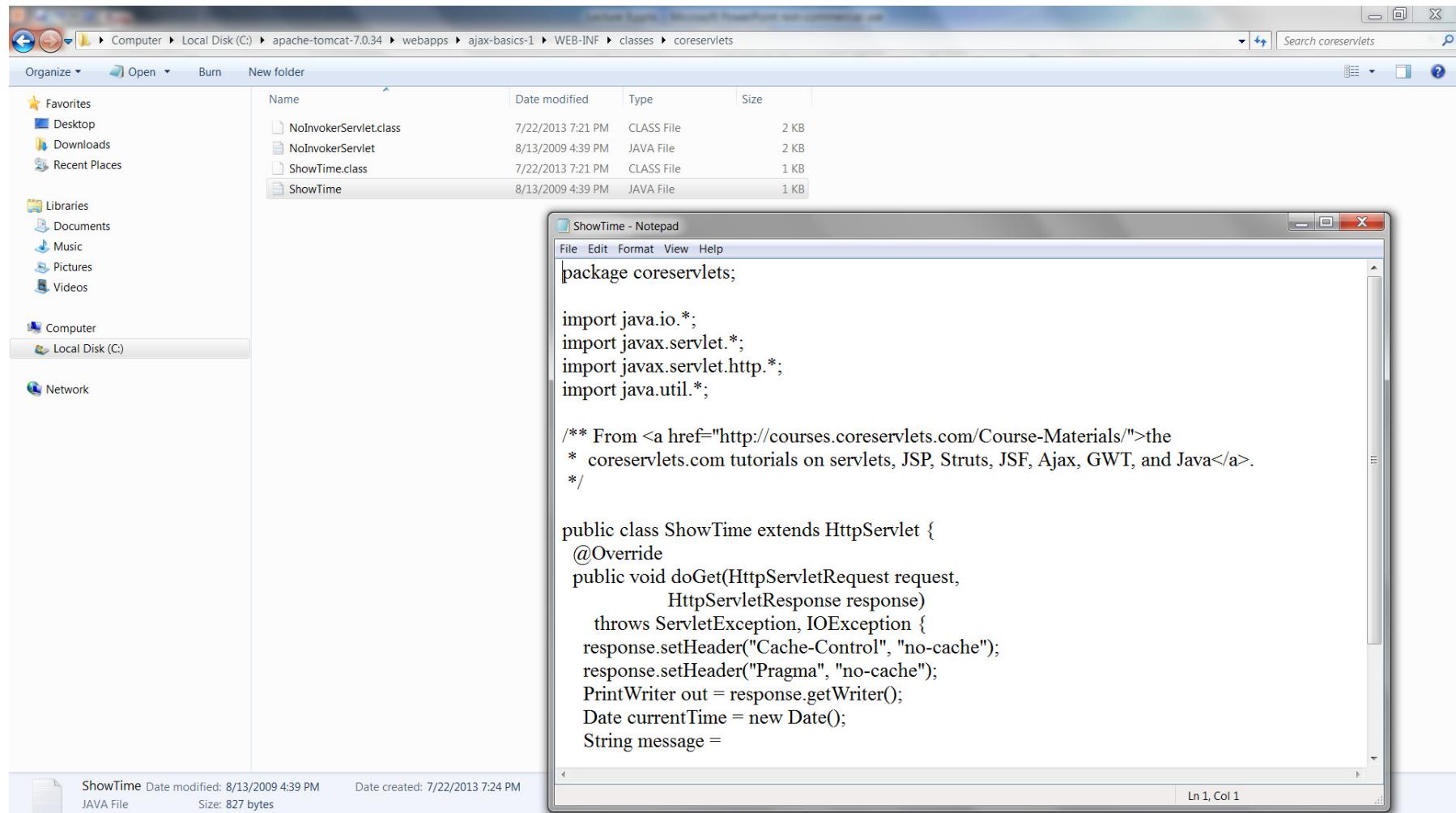
➤ [No changes from previous example.](#)  
[Only address changes, and address comes from the HTML page.](#)

# HTML Code

```
...
<link rel="stylesheet"
      href=".css/styles.css"
      type="text/css"/>
<script src=".scripts/ajax-utils.js"
      type="text/javascript"></script>
...
<fieldset>
  <legend>
    Data from Servlet, Result Shown in Alert Box
  </legend>
  <input type="button" value="Show Server Time"
        onclick='ajaxAlert("show-time")' />
</fieldset>
...
```

➤Address of servlet from @WebServlet (servlets 3.0) or from url-pattern of servlet-mapping in web.xml (servlets 2.5 and earlier).

# Location of the Servlet Code



# Servlet Code

```
package coreservlets;  
import ...  
  
public class ShowTime extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
                      HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setHeader("Cache-Control", "no-cache");  
        response.setHeader("Pragma", "no-cache");  
        PrintWriter out = response.getWriter();  
        Date currentTime = new Date();  
        String message =  
            String.format("It is now %tr on %tD.",  
                         currentTime, currentTime);  
        out.print(message);  
    }  
}
```

# web.xml

```
...
<servlet>
    <servlet-name>ShowTime</servlet-name>
    <servlet-class>coreservlets.ShowTime</servlet-
class>
</servlet>
<servlet-mapping>
    <servlet-name>ShowTime</servlet-name>
    <url-pattern>/show-time</url-pattern>
</servlet-mapping>
...

```

➤ In Tomcat 7 or other servers that support servlets 3.0, you can use `@WebServlet("/show-time")` above the servlet class definition, and omit web.xml entirely.

# Message from Servlet: Results

Ajax Basics: Part I - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Ajax Basics: Part I

localhost/ajax-basics-1/

Secure Search

Ajax Basics: Part I

Data from JSP, Result Shown in Alert Box

Show Server Time

Data from Servlet, Result Shown in Alert Box

Show Server Time

Data from Servlet, Result Shown in HTML

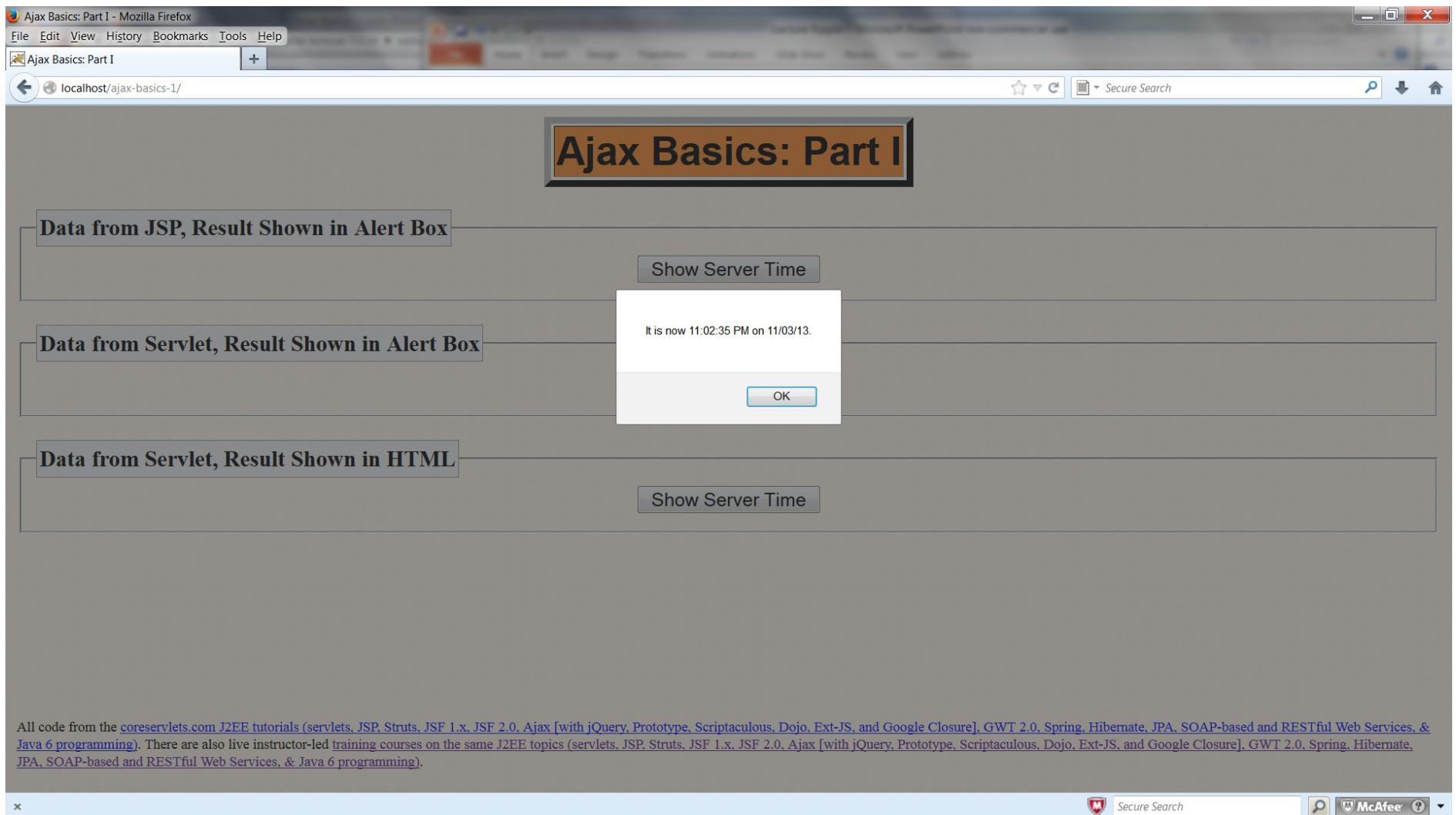
Show Server Time

All code from the [coreservlets.com J2EE tutorials](#) (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming). There are also live instructor-led training courses on the same J2EE topics (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQuery, Prototype, Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate, JPA, SOAP-based and RESTful Web Services, & Java 6 programming).

Secure Search

McAfee

# Message from Servlet: Results

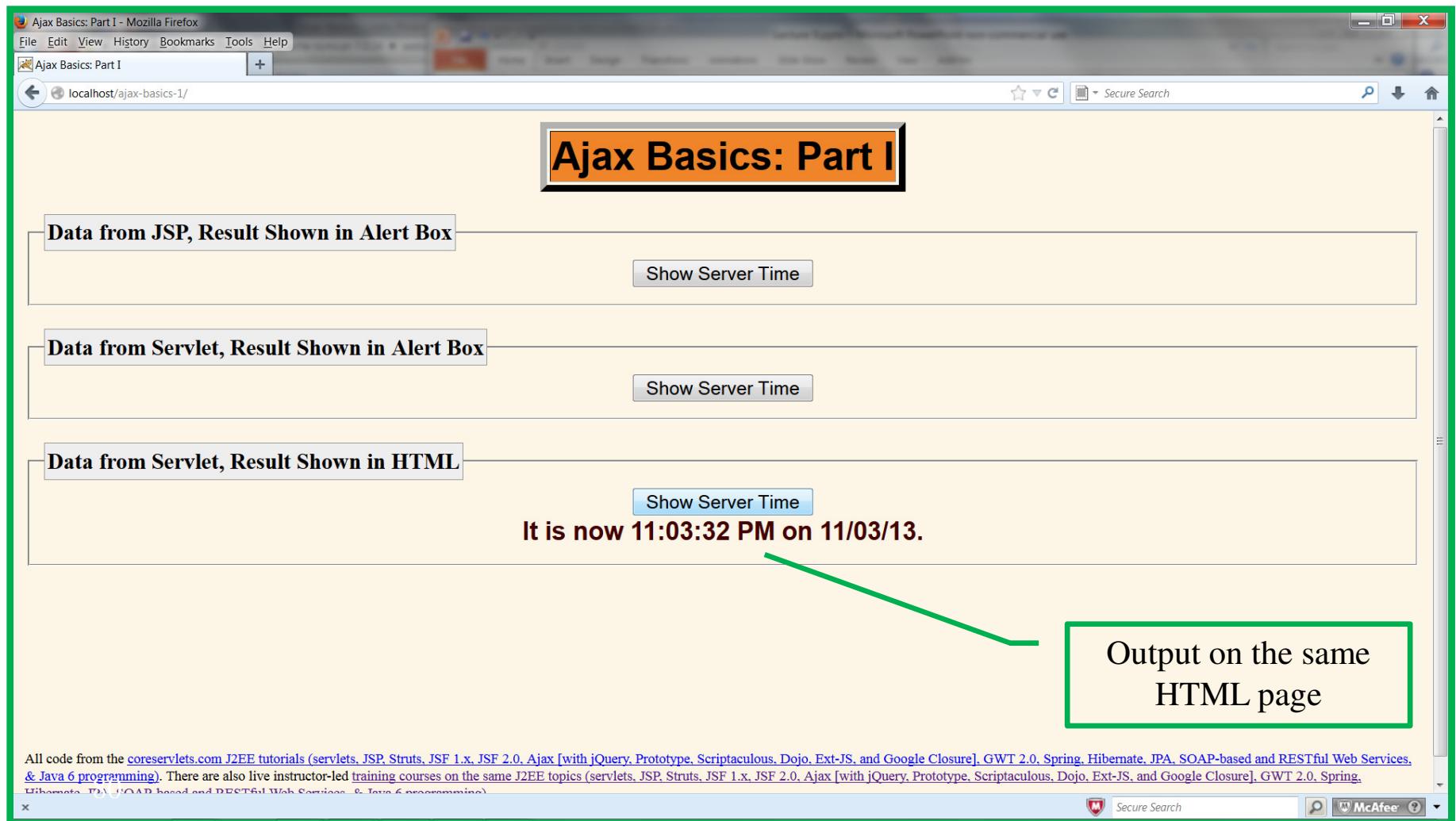


# Building HTML Output

# ShowTime Servlet Example: Design Deficiencies

- Results always shown in dialog (alert) box
  - Alerts usually reserved for errors or warnings
  - Users prefer normal results inside page
  - Solution: use Dynamic HTML to update page with result
    - HTML plus CSS styles represented in the DOM
      - DOM stands for "Document Object Model", an XML view of page
    - JavaScript can insert elements into the DOM
      - Find an element with given id
        - » `someElement = document.getElementById(id);`
      - Insert HTML inside
        - » `someElement.innerHTML = "<h1>blah</h1>";`
    - JavaScript can also read the DOM
      - E.g., look up textfield values (see upcoming example)
        - » `document.getElementById(id).value`

# We want HTML Output on the same page not in a popup dialog box



# Sample Code Snippets for This Approach

index - Notepad

File Edit Format View Help

```
<fieldset>
<legend>Data from JSP, Result Shown in Alert Box</legend>
<input type="button" value="Show Server Time"
       onclick='ajaxAlert("show-time.jsp")'/>
</fieldset>
<p/>
<fieldset>
<legend>Data from Servlet, Result Shown in Alert Box</legend>
<input type="button" value="Show Server Time"
       onclick='ajaxAlert("show-time")'/>
</fieldset>
<p/>
<fieldset>
<legend>Data from Servlet, Result Shown in HTML</legend>
<input type="button" value="Show Server Time"
       onclick='ajaxResult("show-time", "timeResult1")'/>
<div id="timeResult1" class="ajaxResult"></div>
</fieldset>
<p/>
</div>

<br/><br/><br/><br/><br/>
<font size="-3">All code from the
<a href="http://coursescoreservlets.com/Course-Materials/">
coreservlets.com J2EE tutorials (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax
Scriptaculous, Dojo, Ext-JS, and Google Closure], GWT 2.0, Spring, Hibernate
SOAP-based and RESTful Web Services, & Java 6 programming)</a>.
<a href="http://coursescoreservlets.com/">training courses on
the same J2EE topics (servlets, JSP, Struts, JSF 1.x, JSF 2.0, Ajax [with jQu
```

```
ajax-utils - Notepad
File Edit Format View Help

// Make an HTTP request to the given address.
// Display result in the HTML element that has given ID.

function ajaxResult(address, resultRegion) {
    var request = getRequestObject();
    request.onreadystatechange = function() { showResponseText(request, resultRegion); };
    request.open("GET", address, true);
    request.send(null);
}

// Put response text in the HTML element that has given ID.

function showResponseText(request, resultRegion) {
    if ((request.readyState == 4) && (request.status == 200)) {
        htmlInsert(resultRegion, request.responseText);
    }
}

// Insert the html data into the element that has the specified id.

function htmlInsert(id, htmlData) {
    document.getElementById(id).innerHTML = htmlData;
}

// Trick so that the Firebug console.log function will
```

# Dynamically Inserting Text

- HTML
  - <div id="results-placeholder"></div>
- JavaScript
  - resultRegion =  
document.getElementById("results-placeholder");
  - resultRegion.innerHTML = "<h2>Wow!</h2>";
    - For the innerHTML text, you usually use request.responseText or some string derived from request.responseText
- Result after running code
  - <div id="results-placeholder"><h2>Wow!</h2></div>
    - "View source" won't show this.
- Warning
  - Make sure what you insert results in legal XHTML
    - You can't insert block-level elements into inline elements
    - Use correct case for the inserted text

# Summary of New Features

- HTML
  - Define initially blank div element

```
<div id="resultText"></div>
```

- JavaScript response handler
  - Supply an id (resultRegion), find element with that id, and insert response text into innerHTML property

```
document.getElementById(resultRegion).innerHTML =  
    request.responseText;
```

# Steps

- JavaScript
  - Define an object for sending HTTP requests
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
      - Supply as onreadystatechange attribute of request
    - Initiate a GET or POST request to a servlet
    - Send data
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText or responseXML
    - Use innerHTML to insert result into designated element
- HTML
  - Load JavaScript from centralized directory. Use style sheet.
  - Designate control that initiates request
  - Give id to output placeholder region

# Define a Request Object

```
function getRequestObject() {  
    if (window.XMLHttpRequest) {  
        return (new XMLHttpRequest());  
    } else if (window.ActiveXObject) {  
        return (new  
ActiveXObject("Microsoft.XMLHTTP"));  
    } else {  
        ➤No changes from previous examples  
        return (null);  
    }  
}
```

# Initiate Request

```
function ajaxResult(address, resultRegion) {  
    var request = getRequestObject();  
    request.onreadystatechange =  
        function() { showResponseText(request,  
                                      resultRegion); };  
    request.open("GET", address, true);  
    request.send(null);  
}
```

# Handle Response

```
function showResponseText(request, resultRegion)
{
    if ((request.readyState == 4) &&
        (request.status == 200)) {
        htmlInsert(resultRegion,
        request.responseText);
    }
}

function htmlInsert(id, htmlData) {
    document.getElementById(id).innerHTML =
    htmlData;
}
```

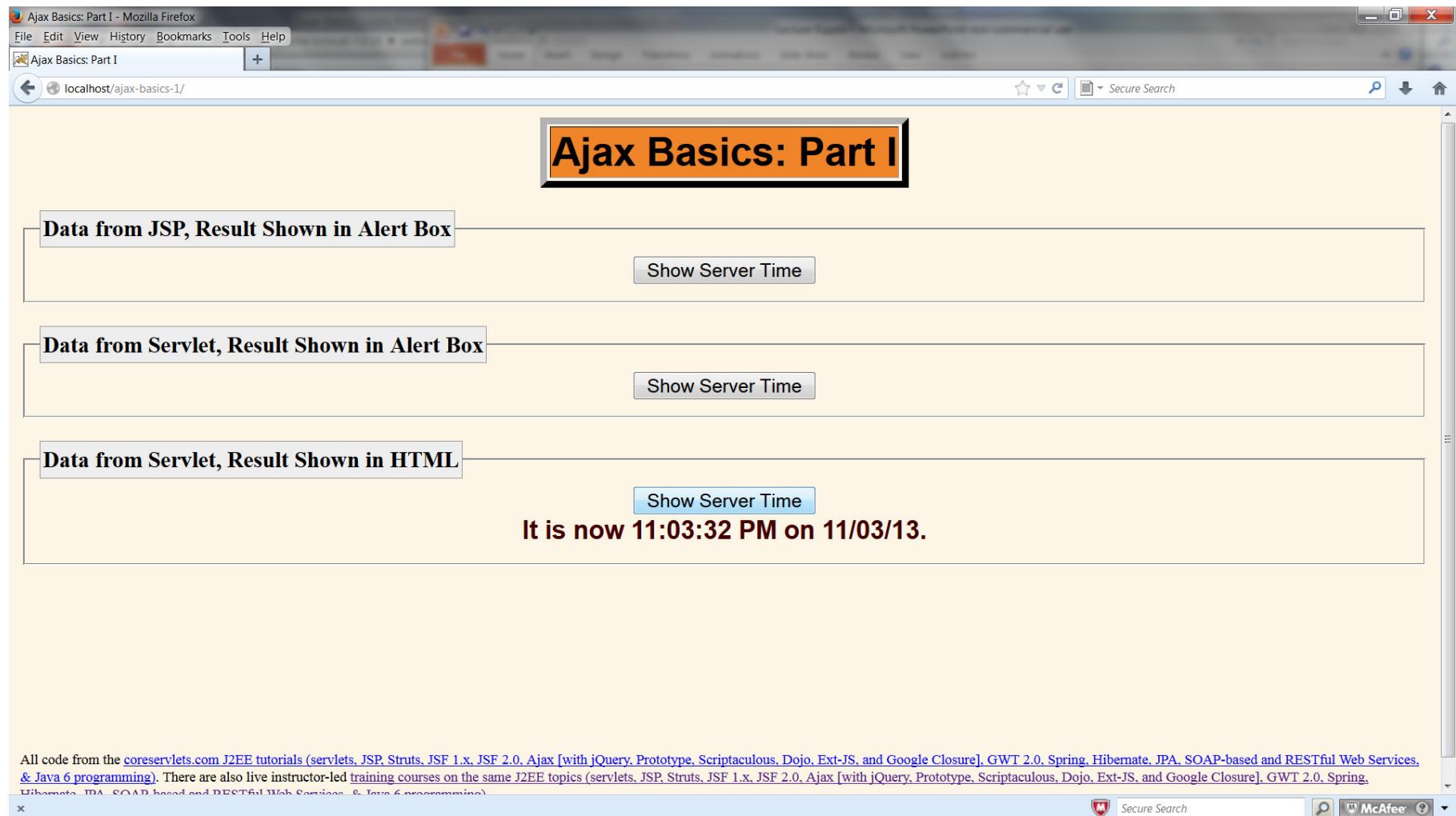
# HTML Code

```
...
<link rel="stylesheet"
      href=".css/styles.css"
      type="text/css"/>
<script src=".scripts/ajax-utils.js"
      type="text/javascript"></script>
...
<fieldset>
  <legend>Data from Servlet, Result Shown in HTML</legend>
  <input type="button" value="Show Server Time"
        onclick='ajaxResult("show-time", "timeResult1")' />
  <div id="timeResult1" class="ajaxResult"></div>
</fieldset>
...
```

# Servlet Code

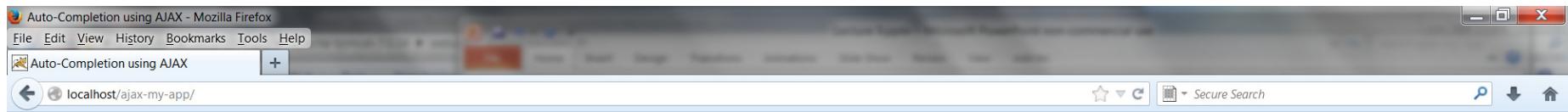
- No changes from previous example
  - Returns a formatted time string

# Building HTML Output: Results



# Auto-Completion using AJAX

- First you get this .... Default file name is index.jsp

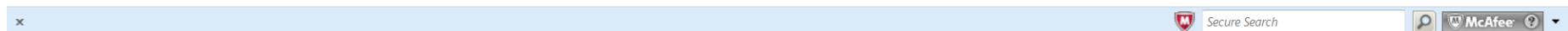


## Auto-Completion using AJAX

This example shows how you can do real time auto-completion using Asynchronous JavaScript and XML (Ajax) interactions.

In the form below enter a name. Possible names that will be completed are displayed below the form. For example, try typing in "Bach," "Mozart," or "Stravinsky," then click on one of the selections to see composer details.

Composer Name:



# Auto-Completion using AJAX

- And as you start typing you get this ....

This screenshot shows a Mozilla Firefox browser window displaying a web page titled "Auto-Completion using AJAX". The URL in the address bar is "localhost/ajax-my-app/index.jsp". The page content describes a real-time auto-completion feature using Asynchronous JavaScript and XML (Ajax) interactions. It instructs the user to enter a name in a form and then click on one of the displayed suggestions to see composer details. A dropdown menu is shown below a text input field containing the letter "B", listing names starting with "B": Bedrich Smetana, Benjamin Britten, Leonard Bernstein, Bela Bartok, Johann Sebastian Bach, Johannes Brahms, and Ludwig van Beethoven.

This example shows how you can do real time auto-completion using Asynchronous JavaScript and XML (Ajax) interactions.

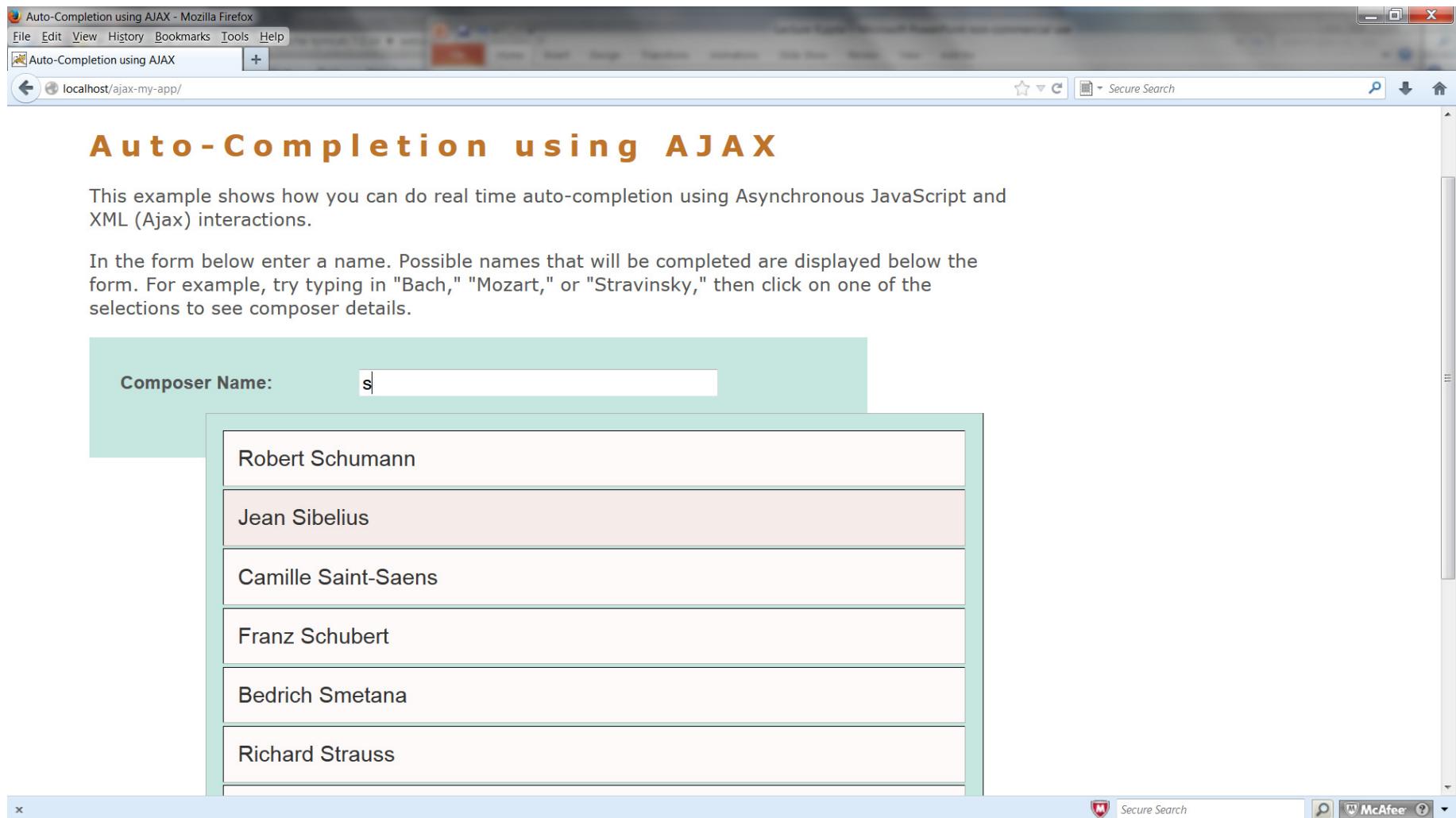
In the form below enter a name. Possible names that will be completed are displayed below the form. For example, try typing in "Bach," "Mozart," or "Stravinsky," then click on one of the selections to see composer details.

Composer Name: B

- Bedrich Smetana
- Benjamin Britten
- Leonard Bernstein
- Bela Bartok
- Johann Sebastian Bach
- Johannes Brahms
- Ludwig van Beethoven

# Auto-Completion using AJAX

- Or this ...



The screenshot shows a Mozilla Firefox browser window with the title "Auto-Completion using AJAX - Mozilla Firefox". The address bar displays "localhost/ajax-my-app/". The main content area has a heading "Auto-Completion using AJAX" followed by a descriptive paragraph. Below the paragraph is a form field labeled "Composer Name:" with the letter "s" typed into it. A dropdown menu is open, listing six names: Robert Schumann, Jean Sibelius, Camille Saint-Saens, Franz Schubert, Bedrich Smetana, and Richard Strauss. The names are displayed in a vertical list with alternating background colors.

**Auto-Completion using AJAX**

This example shows how you can do real time auto-completion using Asynchronous JavaScript and XML (Ajax) interactions.

In the form below enter a name. Possible names that will be completed are displayed below the form. For example, try typing in "Bach," "Mozart," or "Stravinsky," then click on one of the selections to see composer details.

Composer Name: s

- Robert Schumann
- Jean Sibelius
- Camille Saint-Saens
- Franz Schubert
- Bedrich Smetana
- Richard Strauss

# Auto-Completion using AJAX

The screenshot shows a Mozilla Firefox browser window with the title "Auto-Completion using AJAX". The URL in the address bar is "localhost/ajax-my-app/index.jsp". The main content area displays the heading "Auto-Completion using AJAX" and a descriptive text about the example. Below this, there is a form field labeled "Composer Name:" with the letter "s" typed into it. A dropdown menu is displayed, listing several names: Robert Schumann, Jean Sibelius, Camille Saint-Saens, Franz Schubert, Bedrich Smetana, Richard Strauss, and Domenico Scarlatti. The name "Jean Sibelius" is underlined, indicating it is the selected item. A green arrow points from the text "Select this one" to the underlined name. Another green arrow points from the text "Read the URL" to the URL in the browser's address bar, which is "localhost/ajax-my-app/autocomplete?action=lookup&id=36".

This example shows how you can do real time auto-completion using Asynchronous JavaScript and XML (Ajax) interactions.

In the form below enter a name. Possible names that will be completed are displayed below the form. For example, try typing in "Bach," "Mozart," or "Stravinsky," then click on one of the selections to see composer details.

Composer Name: s

- Robert Schumann
- Jean Sibelius
- Camille Saint-Saens
- Franz Schubert
- Bedrich Smetana
- Richard Strauss
- Domenico Scarlatti

localhost/ajax-my-app/autocomplete?action=lookup&id=36

Select this one

Read the URL  
It is a Servlet that will get called

# Auto-Completion using AJAX

Composer Information - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Composer Information

localhost/ajax-my-app/autocomplete?action=lookup&id=36

Secure Search

The Servlet calls the  
composer.jsp

Composer Information

First Name: Jean

Last Name: Sibelius

ID: 36

Category: Romantic

Go back to [application home](#).

Output of  
composer.jsp



# Auto-Completion using AJAX

- Six Files For Auto-Completion Example

The image shows a Windows desktop environment with several open windows:

- index - Notepad**: Contains an HTML snippet for a form with a text input field.
- Javascript - Notepad**: Contains a Java code snippet for a ComposerData class.
- ComposerData - Notepad**: Contains a Java code snippet for a ComposerData class.
- AutoCompleteServlet - Notepad**: Contains a Java code snippet for an AutoCompleteServlet class.
- composer - Notepad**: Contains an HTML snippet for a composer information page.
- Composer - Notepad**: Contains a Java code snippet for a Composer class.
- Composer Information - Mozilla Firefox**: A browser window showing a dropdown menu with options like "Bach", "Arcangelo", and "Corelli".

The code snippets in the Notepad windows are as follows:

- index - Notepad**:

```
<p>JavaScript and XML (Ajax) interactions.</p>

<p>In the form below enter a name. Possible names t<br>the form. For example, try typing in &quot;Bach &lt;<input type="text" size="40" /></td></tr><tbody></tbody></table></form>
```
- Javascript - Notepad**:

```
while (element = targetTop) {
```
- ComposerData - Notepad**:

```
package com.ajax;

import java.util.HashMap;

/**
 *
 * @author nbuser
 */
public class ComposerData {

    private HashMap composers = new HashMap();

    public HashMap getComposers() {
        return composers;
    }

    public ComposerData() {

        composers.put("1", new Composer("1", "Johann Sebastian", "Bach", "Baroque"));
        composers.put("2", new Composer("2", "Arcangelo", "Corelli", "Baroque"));
        composers.put("3", new Composer("3", "George Frideric", "Handel", "Baroque"));
    }
}
```
- AutoCompleteServlet - Notepad**:

```
sb.append("</composition>");
```
- composer - Notepad**:

```
<html>
<head>
<title>Composer Information</title>
<link rel="stylesheet" type="text/css" href="stylesheet.css">
</head>
<body>
```
- Composer - Notepad**:

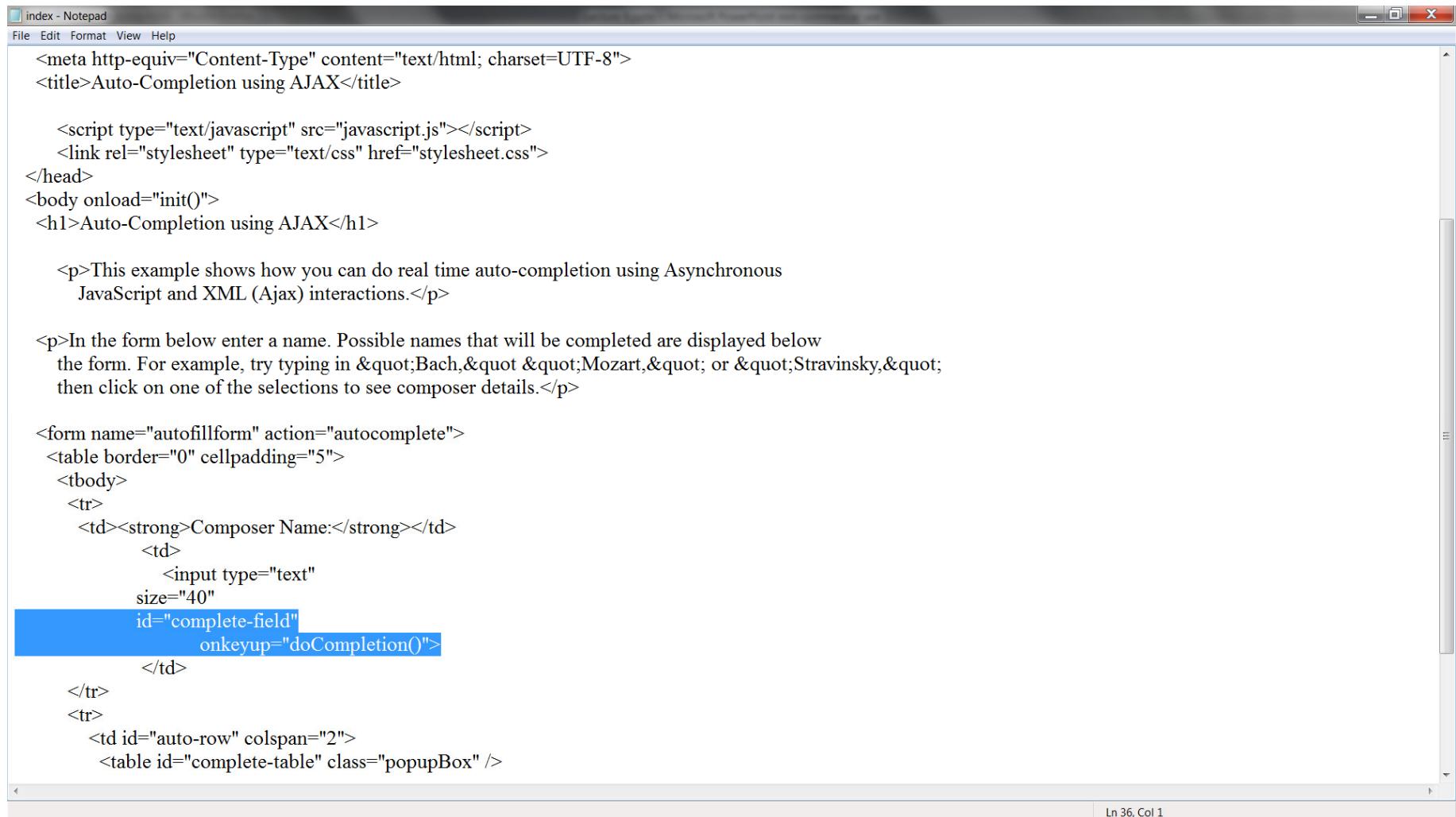
```
package com.ajax;

public class Composer {

    private String id;
    private String firstName;
    private String lastName;
```

# Auto-Completion using AJAX

- index.jsp file



The screenshot shows a Microsoft Notepad window titled "index - Notepad". The window contains the following HTML code:

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Auto-Completion using AJAX</title>

<script type="text/javascript" src="javascript.js"></script>
<link rel="stylesheet" type="text/css" href="stylesheet.css">
</head>
<body onload="init()">
<h1>Auto-Completion using AJAX</h1>

<p>This example shows how you can do real time auto-completion using Asynchronous JavaScript and XML (Ajax) interactions.</p>

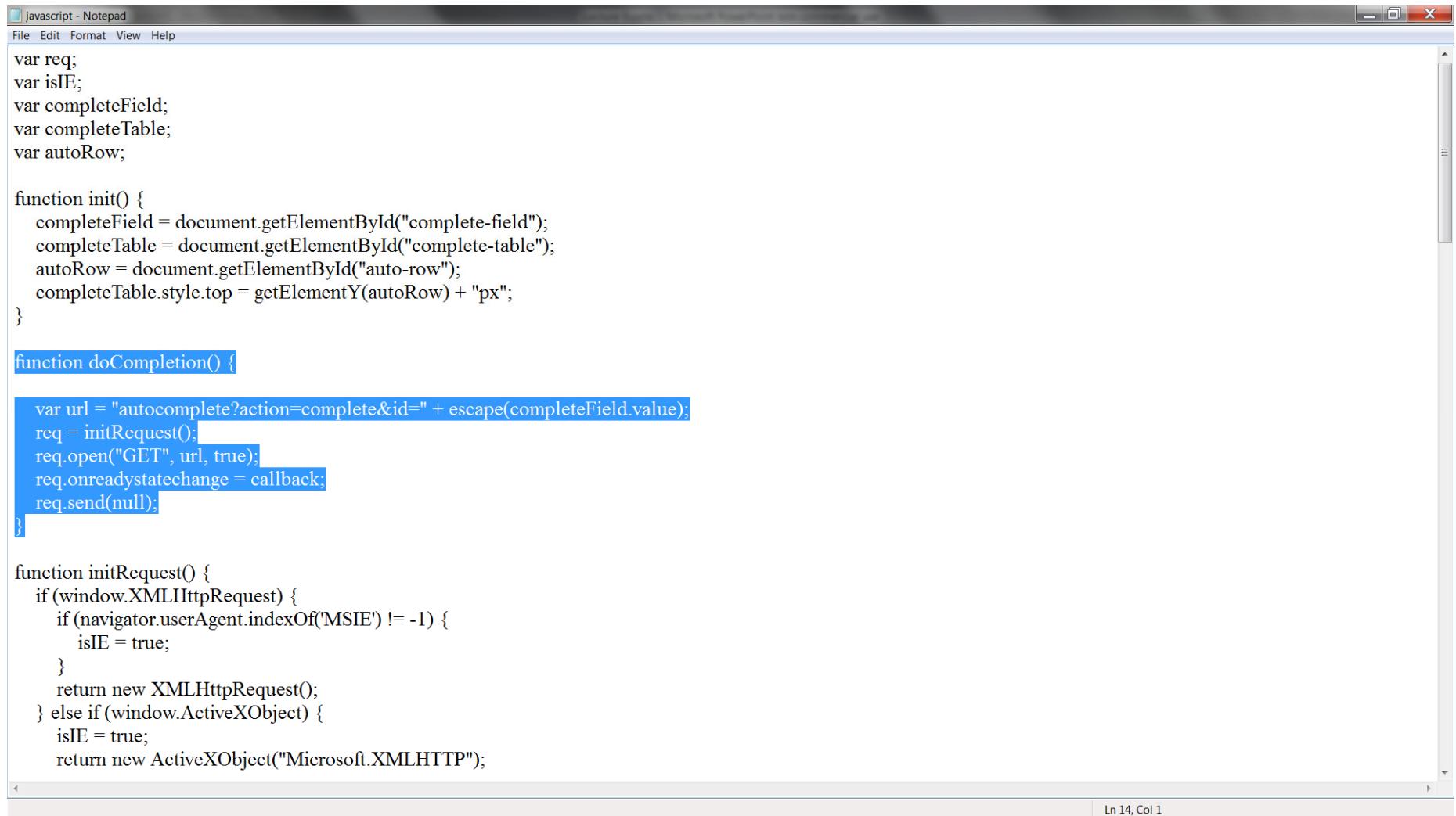
<p>In the form below enter a name. Possible names that will be completed are displayed below the form. For example, try typing in &quot;Bach,&quot;&quot;Mozart,&quot; or &quot;Stravinsky,&quot; then click on one of the selections to see composer details.</p>

<form name="autofillform" action="autocomplete">
<table border="0" cellpadding="5">
<tbody>
<tr>
<td><strong>Composer Name:</strong></td>
<td>
<input type="text"
size="40"
id="complete-field"
onkeyup="doCompletion()"/>
</td>
</tr>
<tr>
<td id="auto-row" colspan="2">
<table id="complete-table" class="popupBox" />
```

The line of code `id="complete-field"` is highlighted with a blue rectangular background.

# Auto-Completion using AJAX

- javascript.js - Complete Action



The screenshot shows a Microsoft Notepad window titled "javascript - Notepad". The code inside the window is as follows:

```
javascript - Notepad
File Edit Format View Help
var req;
var isIE;
var completeField;
var completeTable;
var autoRow;

function init() {
    completeField = document.getElementById("complete-field");
    completeTable = document.getElementById("complete-table");
    autoRow = document.getElementById("auto-row");
    completeTable.style.top = getElementY(autoRow) + "px";
}

function doCompletion() {

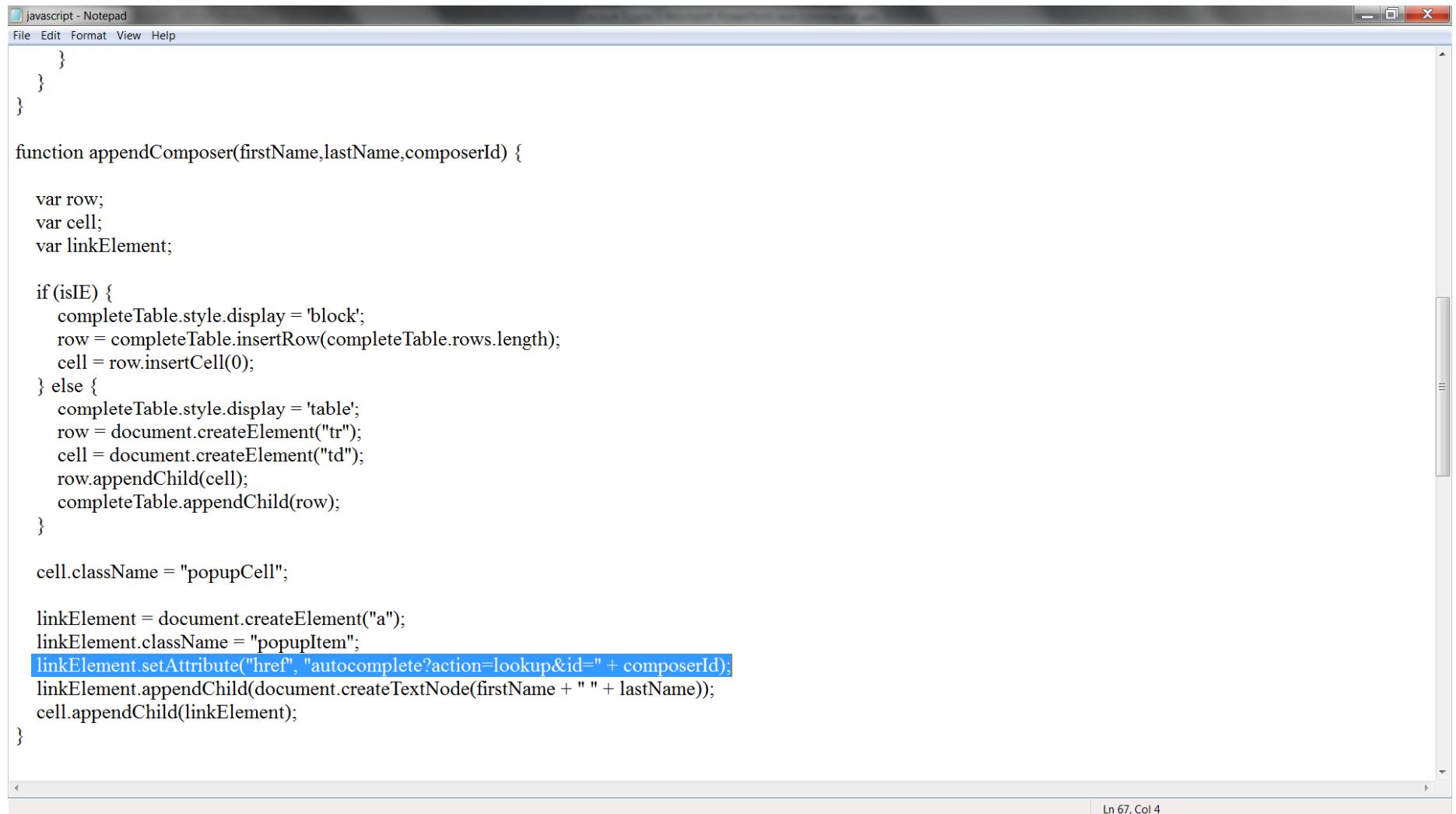
    var url = "autocomplete?action=complete&id=" + escape(completeField.value);
    req = initRequest();
    req.open("GET", url, true);
    req.onreadystatechange = callback;
    req.send(null);
}

function initRequest() {
    if(window.XMLHttpRequest) {
        if(navigator.userAgent.indexOf('MSIE') != -1) {
            isIE = true;
        }
        return new XMLHttpRequest();
    } else if(window.ActiveXObject) {
        isIE = true;
        return new ActiveXObject("Microsoft.XMLHTTP");
    }
}
```

The cursor is positioned at the end of the "req.send(null);" line. The status bar at the bottom right of the Notepad window displays "Ln 14, Col 1".

# Auto-Completion using AJAX

- javascript.js – Lookup Action



A screenshot of a Windows Notepad window titled "javascript - Notepad". The window contains a block of JavaScript code. The code defines a function named "appendComposer" which takes three parameters: "firstName", "lastName", and "composerId". Inside the function, it initializes variables "row", "cell", and "linkElement". It then checks if the browser is Internet Explorer ("isIE"). If so, it sets the style of the "completeTable" to 'block', inserts a new row at the end, and inserts a new cell at index 0. If not, it sets the style to 'table', creates a new "tr" element, creates a new "td" element, appends the "td" to the "tr", and appends the "tr" to the "completeTable". The "cell" is then assigned the class "popupCell". A new "a" element is created and assigned the class "popupItem". Its href attribute is set to "autocomplete?action=lookup&id=" + composerId. It is then appended to the "cell", which is finally appended to the "row". The code ends with a closing brace for the "function" and another for the "appendComposer" function.

```
javascript - Notepad
File Edit Format View Help

}

}

function appendComposer(firstName,lastName,composerId) {
    var row;
    var cell;
    var linkElement;

    if (isIE) {
        completeTable.style.display = 'block';
        row = completeTable.insertRow(completeTable.rows.length);
        cell = row.insertCell(0);
    } else {
        completeTable.style.display = 'table';
        row = document.createElement("tr");
        cell = document.createElement("td");
        row.appendChild(cell);
        completeTable.appendChild(row);
    }

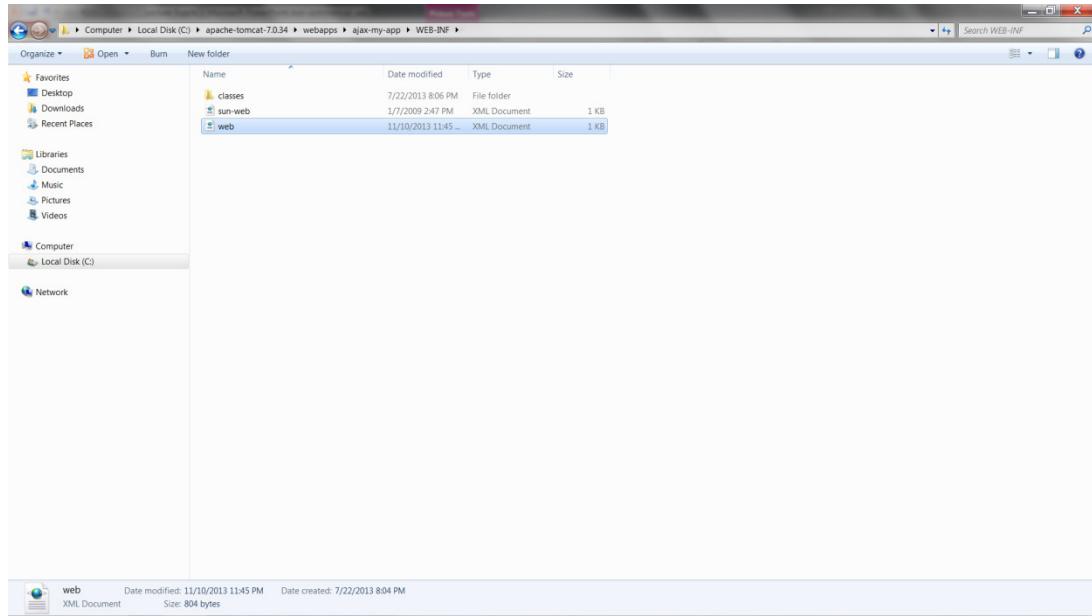
    cell.className = "popupCell";

    linkElement = document.createElement("a");
    linkElement.className = "popupItem";
    linkElement.setAttribute("href", "autocomplete?action=lookup&id=" + composerId);
    linkElement.appendChild(document.createTextNode(firstName + " " + lastName));
    cell.appendChild(linkElement);
}

Ln 67, Col 4
```

# Auto-Completion using AJAX

- Web.xml for Servlet URL mapping

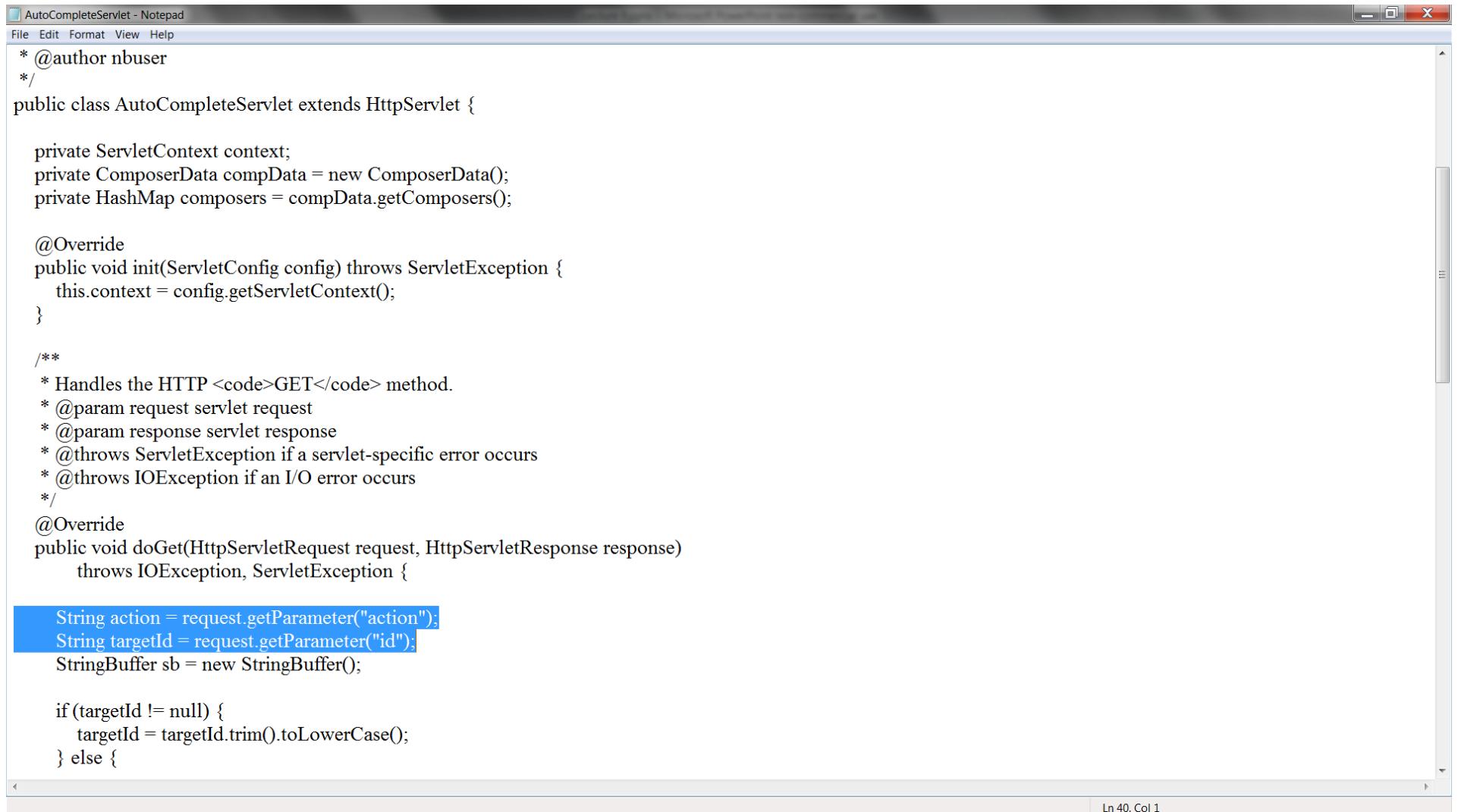


The screenshot shows a Windows file explorer window titled "Computer" with the path "Local Disk (C) > apache-tomcat-7.0.34 > webapps > ajax-my-app > WEB-INF". Inside the "WEB-INF" folder, there are three files: "classes", "sun-web.xml", and "web.xml". The "web.xml" file is selected and its properties are shown at the bottom: "Name: web", "Date modified: 11/10/2013 11:45 PM", "Date created: 7/22/2013 8:04 PM", and "Size: 804 bytes".

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xn
<servlet>
    <servlet-name>AutoCompleteServlet</servlet-name>
    <servlet-class>com.ajax.AutoCompleteServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>AutoCompleteServlet</servlet-name>
    <url-pattern>/autocomplete</url-pattern>
</servlet-mapping>
<session-config>
    <session-timeout>
        30
    </session-timeout>
</session-config>
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

# Auto-Completion using AJAX

- AutoCompleteServlet



The screenshot shows a Windows Notepad window titled "AutoCompleteServlet - Notepad". The window contains Java code for an AutoCompleteServlet. The code includes annotations like @author nbuser, @Override, and various Javadoc-style comments. A blue rectangular highlight is placed over the first few lines of the code, specifically around the class definition and some initial variable declarations. The status bar at the bottom right of the Notepad window displays "Ln 40, Col 1".

```
* @author nbuser
*/
public class AutoCompleteServlet extends HttpServlet {

    private ServletContext context;
    private ComposerData compData = new ComposerData();
    private HashMap composers = compData.getComposers();

    @Override
    public void init(ServletConfig config) throws ServletException {
        this.context = config.getServletContext();
    }

    /**
     * Handles the HTTP <code>GET</code> method.
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {

        String action = request.getParameter("action");
        String targetId = request.getParameter("id");
        StringBuffer sb = new StringBuffer();

        if (targetId != null) {
            targetId = targetId.trim().toLowerCase();
        } else {
```

# Auto-Completion using AJAX

- AutoCompleteServlet

```
AutoCompleteServlet - Notepad
File Edit Format View Help

boolean namesAdded = false;
if (action.equals("complete")) {

    // check if user sent empty string
    if (!targetId.equals("")) {

        Iterator it = composers.keySet().iterator();

        while (it.hasNext()) {
            String id = (String) it.next();
            Composer composer = (Composer) composers.get(id);

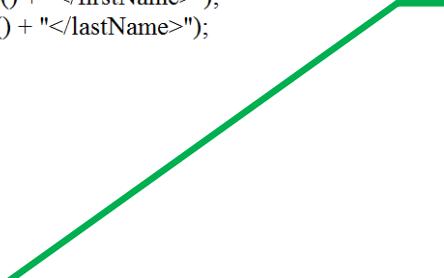
            if ( // targetId matches first name
                composer.getFirstName().toLowerCase().startsWith(targetId) ||
                // targetId matches last name
                composer.getLastName().toLowerCase().startsWith(targetId) ||
                // targetId matches full name
                composer.getFirstName().toLowerCase().concat(" ")
                    .concat(composer.getLastName().toLowerCase()).startsWith(targetId)) {

                sb.append("<composer>");
                sb.append("<id>" + composer.getId() + "</id>");
                sb.append("<firstName>" + composer.getFirstName() + "</firstName>");
                sb.append("<lastName>" + composer.getLastName() + "</lastName>");
                sb.append("</composer>");
                namesAdded = true;
            }
        }
    }
}
```

XML data added to StringBuffer

# Auto-Completion using AJAX

- AutoCompleteServlet



```
AutoCompleteServlet - Notepad
File Edit Format View Help
// targetId matches full name
composer.getFirstName().toLowerCase().concat(" ")
.concat(composer.getLastName().toLowerCase()).startsWith(targetId)) {

    sb.append("<composer>");
    sb.append("<id>" + composer.getId() + "</id>");
    sb.append("<firstName>" + composer.getFirstName() + "</firstName>");
    sb.append("<lastName>" + composer.getLastName() + "</lastName>");
    sb.append("</composer>");
    namesAdded = true;
}
}

if (namesAdded) {
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("<composers>" + sb.toString() + "</composers>");
} else {
    //nothing to show
    response.setStatus(HttpServletResponse.SC_NO_CONTENT);
}

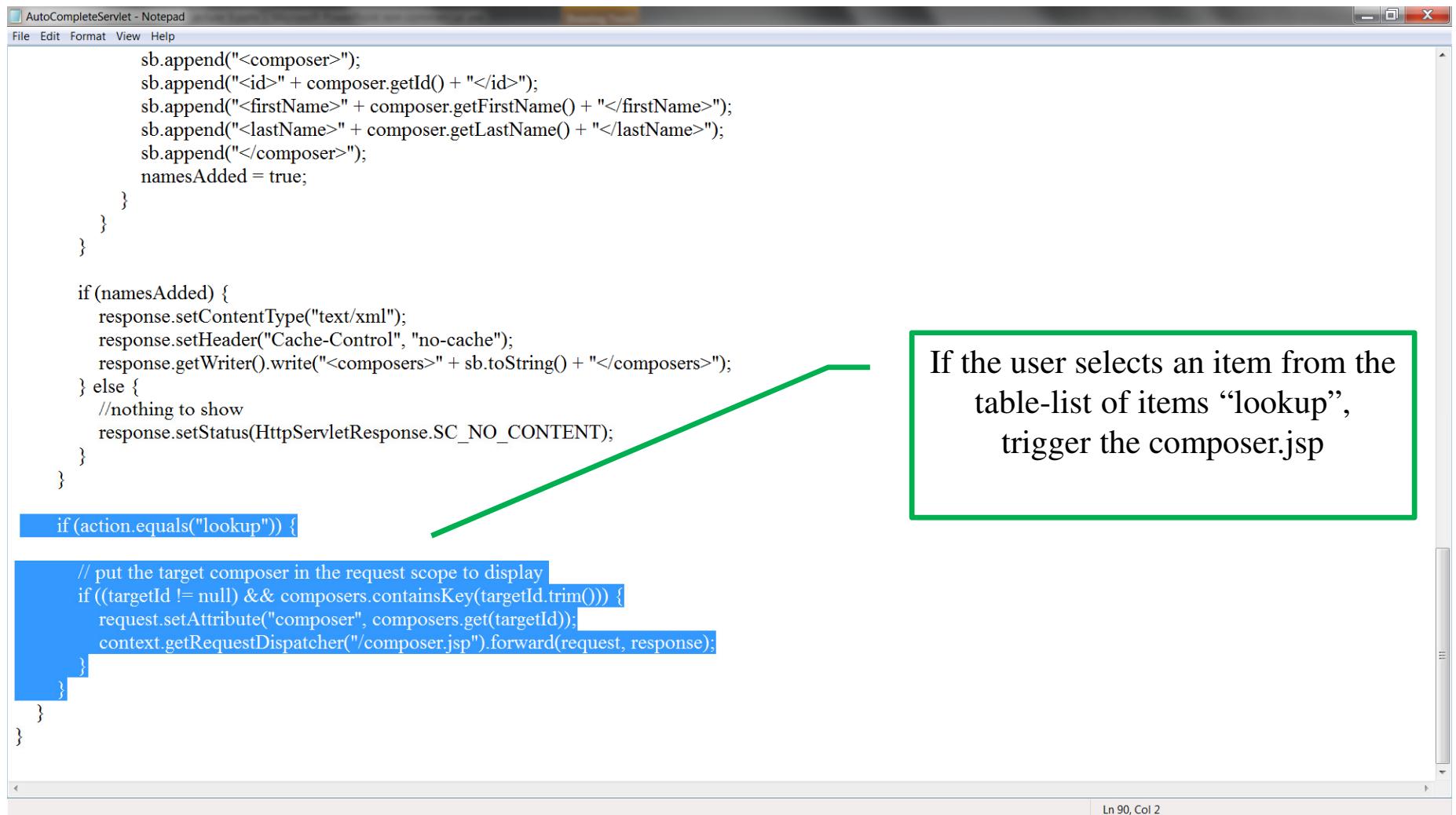
if (action.equals("lookup")) {

    // put the target composer in the request scope to display
    if ((targetId != null) && composers.containsKey(targetId.trim())) {
        request.setAttribute("composer", composers.get(targetId));
        context.getRequestDispatcher("/composer.jsp").forward(request, response);
    }
}

Write XML data to Response object
Ln 80, Col 1
```

# Auto-Completion using AJAX

- AutoCompleteServlet



The screenshot shows a Notepad window titled "AutoCompleteServlet - Notepad". The code is a Java servlet implementation:

```
sb.append("<composer>");
sb.append("<id>" + composer.getId() + "</id>");
sb.append("<firstName>" + composer.getFirstName() + "</firstName>");
sb.append("<lastName>" + composer.getLastName() + "</lastName>");
sb.append("</composer>");
namesAdded = true;
}
}
}

if (namesAdded) {
    response.setContentType("text/xml");
    response.setHeader("Cache-Control", "no-cache");
    response.getWriter().write("<composers>" + sb.toString() + "</composers>");
} else {
    //nothing to show
    response.setStatus(HttpServletResponse.SC_NO_CONTENT);
}

if (action.equals("lookup")) {

    // put the target composer in the request scope to display
    if ((targetId != null) && composers.containsKey(targetId.trim())) {
        request.setAttribute("composer", composers.get(targetId));
        context.getRequestDispatcher("/composer.jsp").forward(request, response);
    }
}
}
```

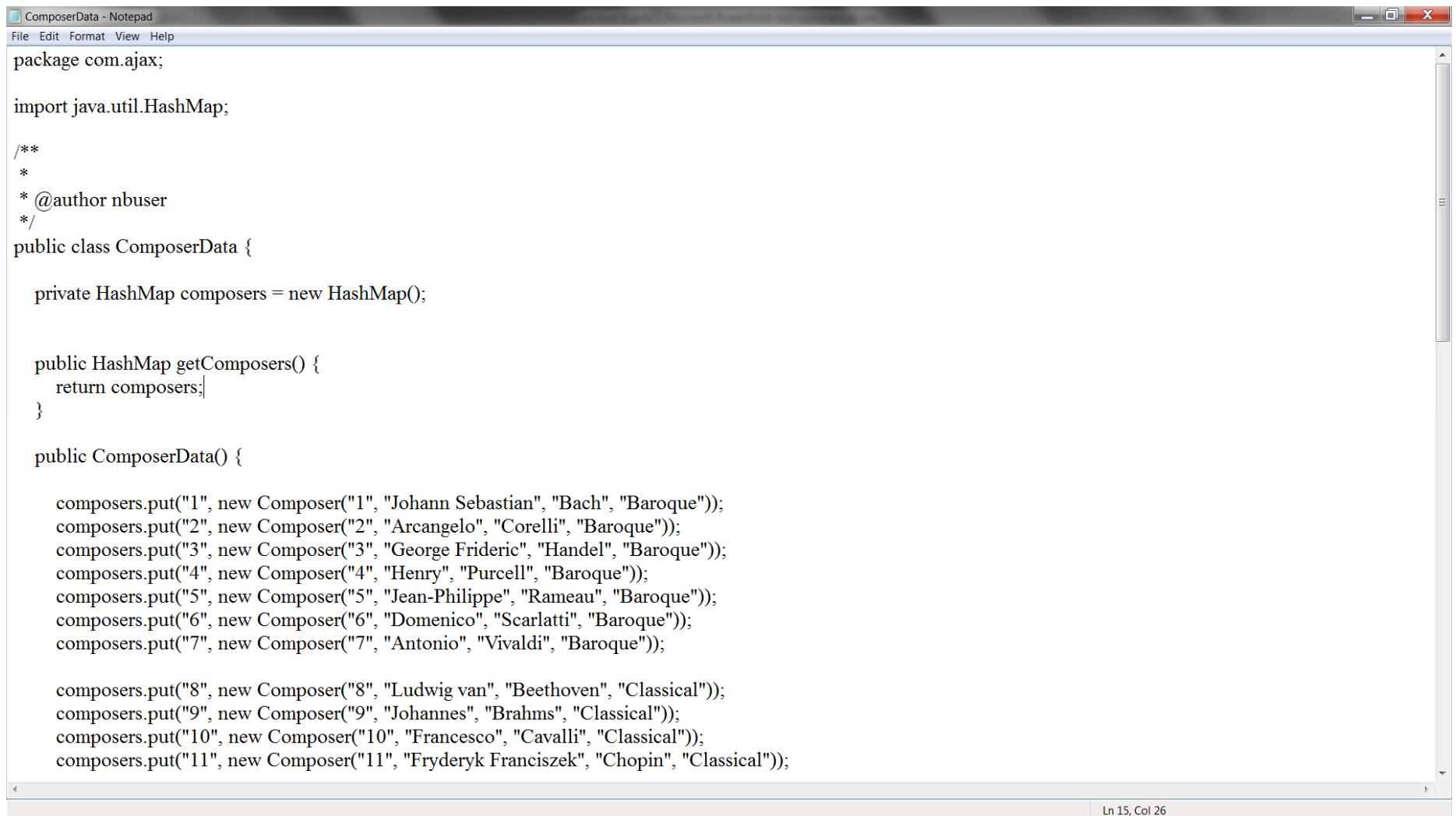
A green arrow points from the text "If the user selects an item from the table-list of items “lookup”, trigger the composer.jsp" to the line "if (action.equals("lookup")) {".

If the user selects an item from the table-list of items “lookup”, trigger the composer.jsp

Ln 90, Col 2

# Auto-Completion using AJAX

- ComposerData Class



The screenshot shows a Windows Notepad window titled "ComposerData - Notepad". The window contains Java code for a class named "ComposerData". The code includes imports for "com.ajax" and "java.util.HashMap", a class definition with a constructor and a method "getComposers()", and a list of composers stored in a HashMap. The cursor is positioned at the end of the "return" statement in the "getComposers()" method.

```
package com.ajax;

import java.util.HashMap;

/**
 *
 * @author nbuser
 */
public class ComposerData {

    private HashMap composers = new HashMap();

    public HashMap getComposers() {
        return composers;
    }

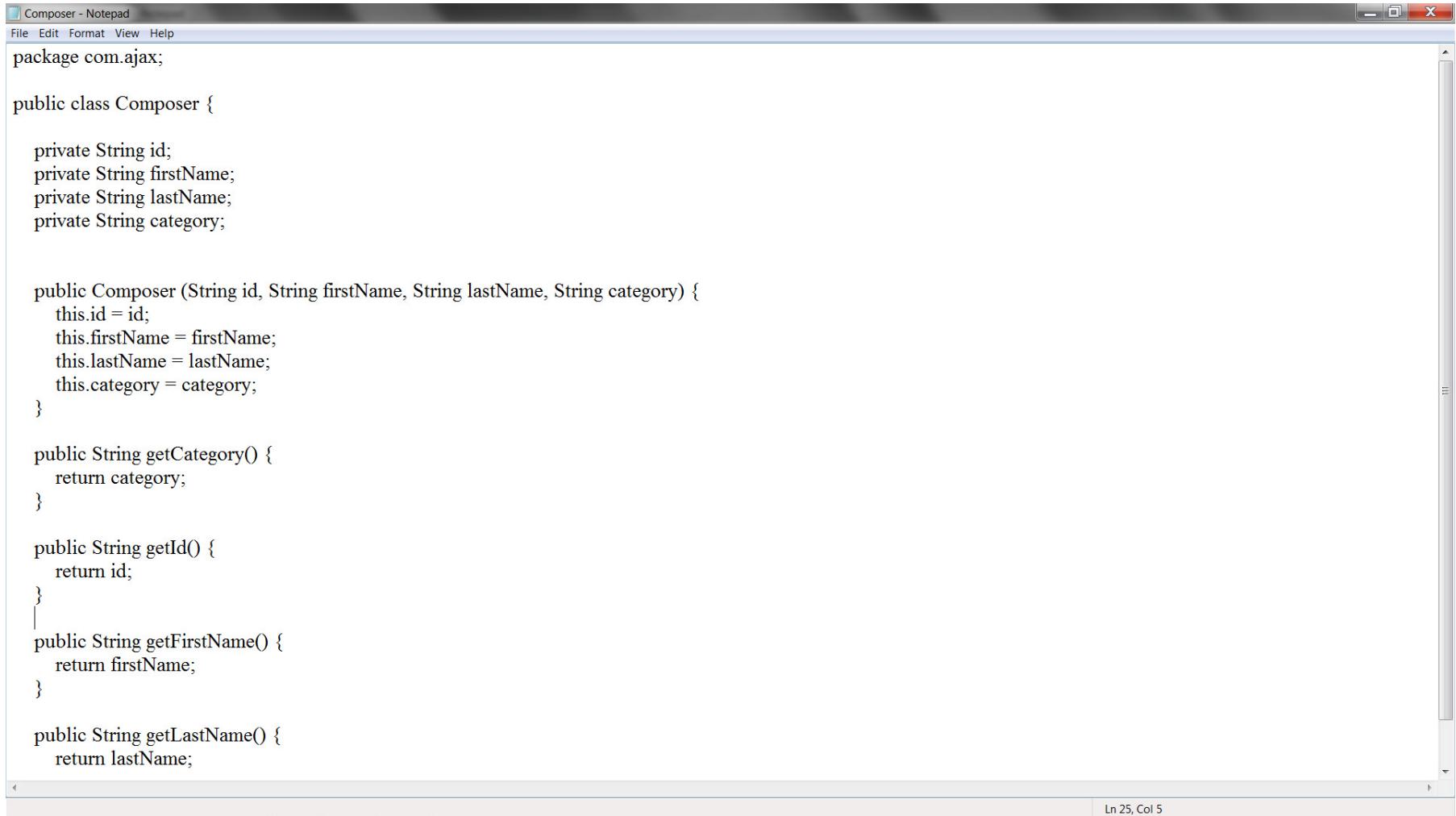
    public ComposerData() {

        composers.put("1", new Composer("1", "Johann Sebastian", "Bach", "Baroque"));
        composers.put("2", new Composer("2", "Arcangelo", "Corelli", "Baroque"));
        composers.put("3", new Composer("3", "George Frideric", "Handel", "Baroque"));
        composers.put("4", new Composer("4", "Henry", "Purcell", "Baroque"));
        composers.put("5", new Composer("5", "Jean-Philippe", "Rameau", "Baroque"));
        composers.put("6", new Composer("6", "Domenico", "Scarlatti", "Baroque"));
        composers.put("7", new Composer("7", "Antonio", "Vivaldi", "Baroque"));

        composers.put("8", new Composer("8", "Ludwig van", "Beethoven", "Classical"));
        composers.put("9", new Composer("9", "Johannes", "Brahms", "Classical"));
        composers.put("10", new Composer("10", "Francesco", "Cavalli", "Classical"));
        composers.put("11", new Composer("11", "Fryderyk Franciszek", "Chopin", "Classical"));
    }
}
```

# Auto-Completion using AJAX

- Composer Class



The screenshot shows a Windows Notepad window titled "Composer - Notepad". The window contains Java code for a "Composer" class. The code includes private fields for id, firstName, lastName, and category, a constructor that initializes these fields, and four getter methods (getCategory, getId, getFirstName, and getLastName). The Notepad interface has a menu bar with File, Edit, Format, View, and Help, and a status bar at the bottom indicating "Ln 25, Col 5".

```
package com.ajax;

public class Composer {

    private String id;
    private String firstName;
    private String lastName;
    private String category;

    public Composer (String id, String firstName, String lastName, String category) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
        this.category = category;
    }

    public String getCategory() {
        return category;
    }

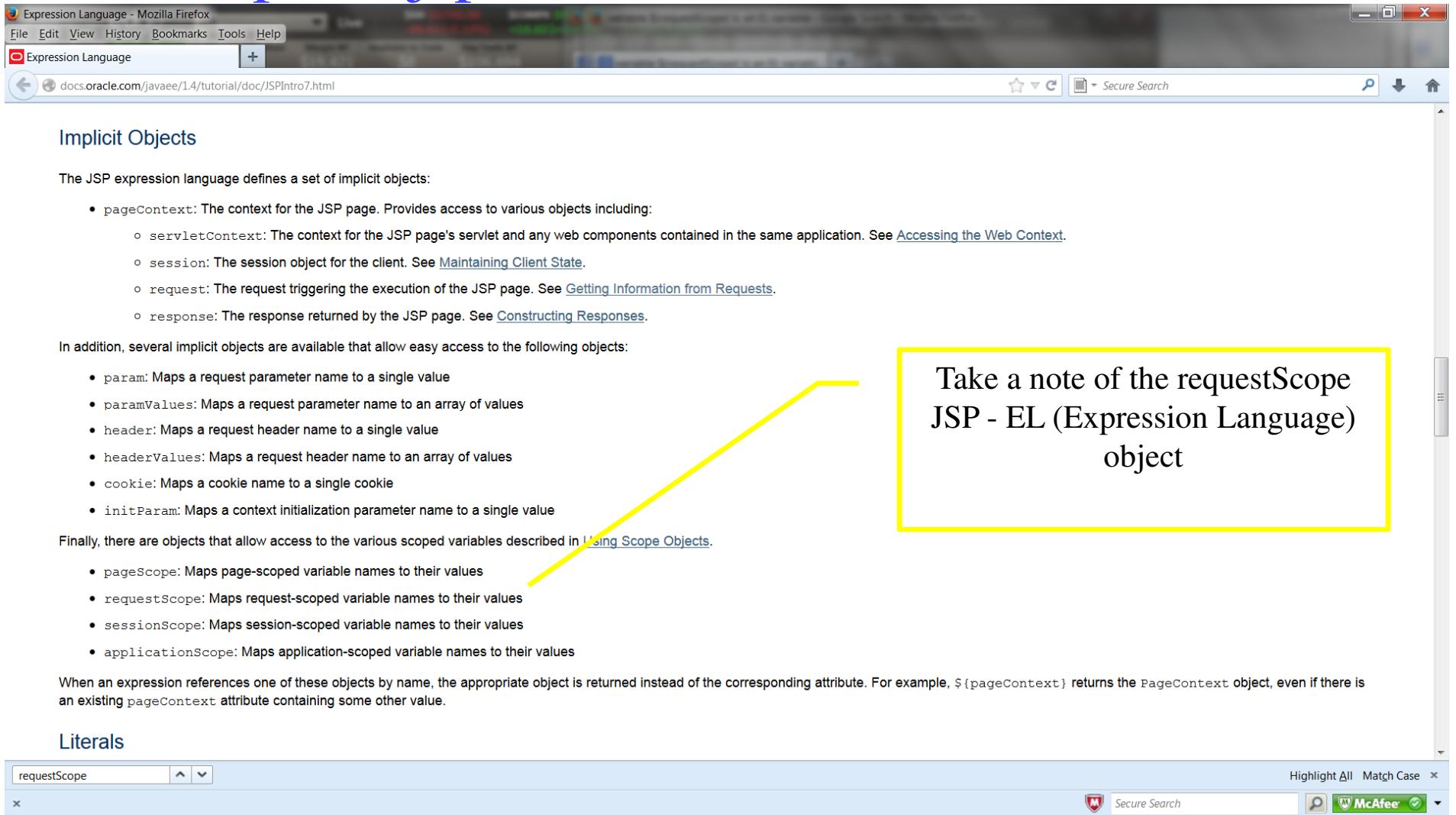
    public String getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```

# Auto-Completion using AJAX

- Composer.jsp



The screenshot shows a Mozilla Firefox browser window with the title "Expression Language - Mozilla Firefox". The address bar contains "docs.oracle.com/javaee/1.4/tutorial/doc/JSPIntro7.html". The page content is titled "Implicit Objects". It describes the JSP expression language's set of implicit objects, including `pageContext`, `servletContext`, `session`, `request`, and `response`. It also lists several implicit objects for easy access to request parameters, headers, cookies, and initialization parameters. A note highlights the `requestScope` object. The bottom section discusses scoped variables and their corresponding objects like `pageScope`, `requestScope`, `sessionScope`, and `applicationScope`. A search bar at the bottom is set to "requestScope".

**Implicit Objects**

The JSP expression language defines a set of implicit objects:

- `pageContext`: The context for the JSP page. Provides access to various objects including:
  - `servletContext`: The context for the JSP page's servlet and any web components contained in the same application. See [Accessing the Web Context](#).
  - `session`: The session object for the client. See [Maintaining Client State](#).
  - `request`: The request triggering the execution of the JSP page. See [Getting Information from Requests](#).
  - `response`: The response returned by the JSP page. See [Constructing Responses](#).

In addition, several implicit objects are available that allow easy access to the following objects:

- `param`: Maps a request parameter name to a single value
- `paramValues`: Maps a request parameter name to an array of values
- `header`: Maps a request header name to a single value
- `headerValues`: Maps a request header name to an array of values
- `cookie`: Maps a cookie name to a single cookie
- `initParam`: Maps a context initialization parameter name to a single value

Finally, there are objects that allow access to the various scoped variables described in [Using Scope Objects](#).

- `pageScope`: Maps page-scoped variable names to their values
- `requestScope`: Maps request-scoped variable names to their values
- `sessionScope`: Maps session-scoped variable names to their values
- `applicationScope`: Maps application-scoped variable names to their values

When an expression references one of these objects by name, the appropriate object is returned instead of the corresponding attribute. For example,  `${pageContext}`  returns the `PageContext` object, even if there is an existing `pageContext` attribute containing some other value.

**Literals**

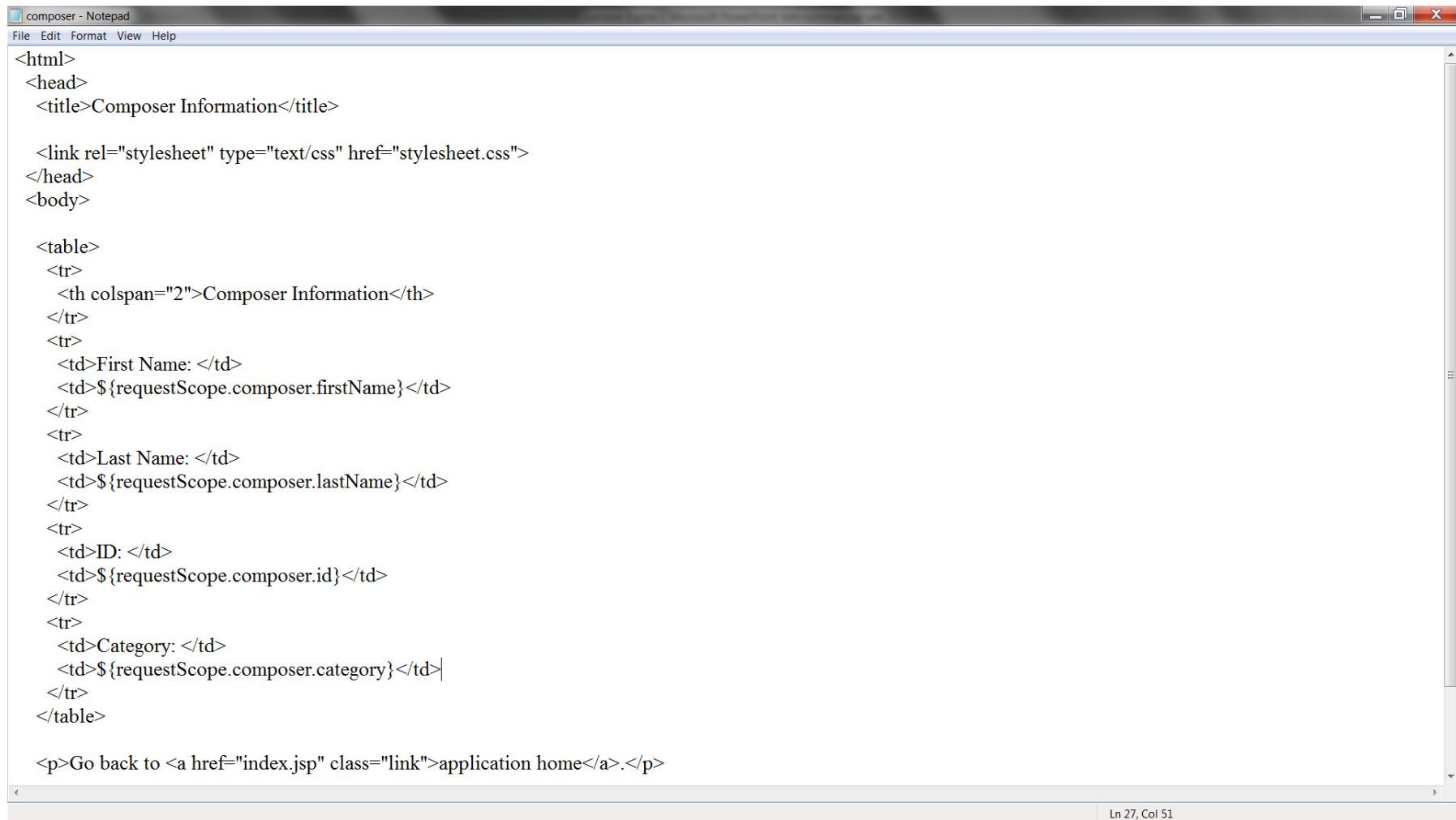
requestScope

Highlight All Match Case ×

Secure Search McAfee ✓

# Auto-Completion using AJAX

- Composer.jsp



The screenshot shows a Windows Notepad window titled "composer - Notepad". The window contains JSP code for a page named "Composer Information". The code includes an HTML header with a title, a CSS link, and a table with five rows. The first row has a colspan of 2. The subsequent rows contain form fields for First Name, Last Name, ID, and Category, each paired with a value from the request scope. A final paragraph at the bottom provides a link back to the index page.

```
<html>
<head>
<title>Composer Information</title>

<link rel="stylesheet" type="text/css" href="stylesheet.css">
</head>
<body>

<table>
<tr>
<th colspan="2">Composer Information</th>
</tr>
<tr>
<td>First Name:</td>
<td>${requestScope.composer.firstName}</td>
</tr>
<tr>
<td>Last Name:</td>
<td>${requestScope.composer.lastName}</td>
</tr>
<tr>
<td>ID:</td>
<td>${requestScope.composer.id}</td>
</tr>
<tr>
<td>Category:</td>
<td>${requestScope.composer.category}</td>
</tr>
</table>

<p>Go back to <a href="index.jsp" class="link">application home</a>.</p>
```

Ln 27, Col 51

# Summary

- JavaScript
  - Define request object
    - Check for both Microsoft and non-MS objects. Identical code in all apps.
  - Initiate request
    - Get request object
    - Designate an anonymous response handler function
    - Initiate a GET request with relative URL
  - Handle response
    - Wait for readyState of 4 and HTTP status of 200
    - Extract return text with responseText
    - Do something with result
      - Use innerHTML to insert result into designated element
- HTML
  - Give id to placeholder (often a div). Initiate process on user action.
- Java
  - Use JSP, servlet, or combination (MVC) as appropriate.
  - Prevent browser caching.