jQuery and Ajax

- Introduction to Ajax
- Ajax in Raw JavaScript
- jQuery and Ajax

What is AJAX?

- AJAX (Asychronous JavaScript and XML) uses a combination of existing technologies:
  - JavaScript
  - The XMLHttpRequest Object
  - HTML Document Object Model
- AJAX lets you create Web applications with a higher degree of interactivity

Ajax

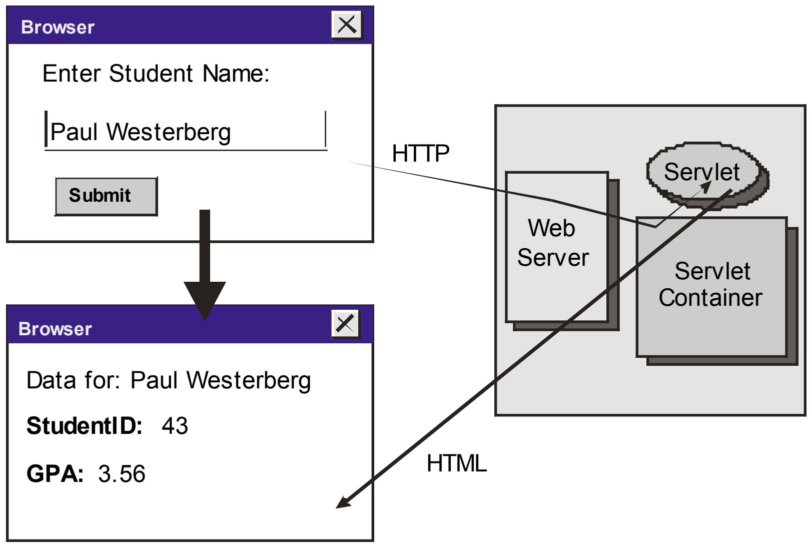
The notion of asynchronous request processing using the XMLHttpRequest object has been around for several years, but the term "AJAX" was coined by Jesse James Garrett of Adaptive Path.

You can read his essay at www.adaptivepath.com/publications/essays/archives/000385.php.

Note that to use AJAX, you're not required to use XML.

Traditional Web Processing

- In the traditional model, users submit pages and Web applications generate complete pages
- This can be inefficient, especially if most of the page is unchanged from one request to the next

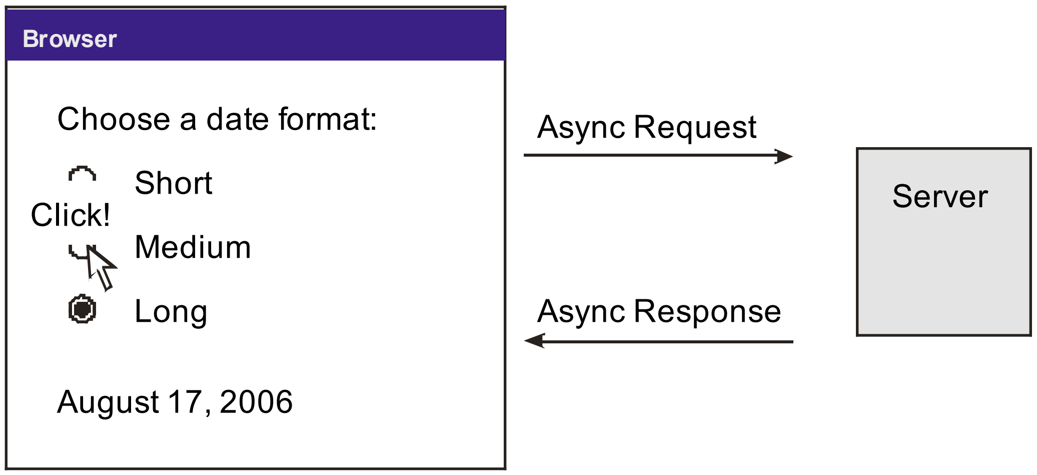


The Web started as a classic two-tier, client-server system in which the Web browser client is nothing more than a display device. All of the processing power and page-generation logic resides on the server.

While this model has been fantastically successful, it's not necessarily the correct fit for all types of applications.

AJAX Processing

- With AJAX, **any** user action can trigger a request to the server for more or different data
- Since the request is asynchronous, the user can keep working while the request is processed



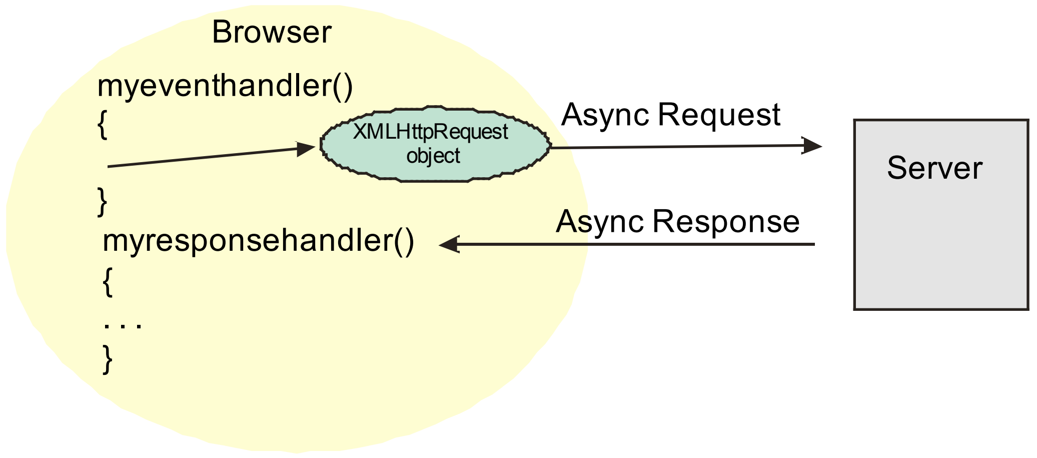
In an AJAX Web application, JavaScript on the client side can send requests without requiring the user to press a Submit button. In this figure, we show an application that updates a date display whenever the user clicks a radio button. In response, the server sends back a differently formatted string containing the current date.

AJAX Applications

- Perhaps the first AJAX application was Microsoft's Outlook Web Access
- Here is a partial listing of others:
    - Flickr
    - Google Maps
    - Google Mail
    - Yahoo Mail

The XMLHttpRequest Object

- The **XMLHttpRequest** object was first implemented by Microsoft Internet Explorer, but is now a part of other browsers such as Firefox and Opera
- This object defines an API for making remote requests, either synchronously or asynchronously



The XMLHttpRequest object is the foundation of AJAX.

This object lets a JavaScript event handler (e.g. from a radio-button click), send a request to the server and establish another handler to process the response. The response handler can use the DOM to modify the page content based on the returned data. This technique is what lets us write AJAX applications that perform requests without requiring a full page reload.

The XMLHttpRequest object has an interesting history. First implemented by Microsoft Internet Explorer, it has since been "ported" to other browsers and has become a standard part of modern browsers.

XMLHttpRequest: Raw JavaScript

- Retrieving the XMLHttpRequest object reference is somewhat browser dependent

```
function getXMLHttpRequest()
{
  var xRequest=null;
  // handle non-Microsoft browsers
  if (window.XMLHttpRequest)
  {
    xRequest=new XMLHttpRequest();
  }
  // handle Microsoft browsers
  else if (typeof ActiveXObject != "undefined")
  {
    xRequest=new ActiveXObject("Microsoft.XMLHTTP");
  }
  return xRequest;
}
```

While the XMLHttpRequest object itself is mostly standard across browsers, the act of creating the object is different.

Here we show a simple JavaScript method that returns a reference to the object on different browsers.

Sending a Request: Raw JavaScript

- The XMLHttpRequest object supports both HTTP **GET** and **POST** requests
- Here we show using a **GET** with parameters and an asychronous *callback handler*

```
1    var req;
2    function sendAsyncGetRequest(url)
3    {
4      req = getXMLHttpRequest();
5      if (req)
6      {
7        req.open("GET", url, true);
8        req.onreadystatechange = callback;
9        req.send(null);
10     }
11   }
```

```
var url =
  "MyServlet?myparm='abc'";
sendAsyncGetRequest(url);
```

This code listing shows defining "req" as a global JavaScript variable and then a method that sends an asynchronous "get" request to the server.

In line 4, we call the method shown on the last page that creates the XMLHttpRequest object. Assuming that works, we then open a request using "get", the passed URL and specify "true" to configure an asynchronous request. Then in line 8, we register our asynchronous response handler (code shown on the next page). In line 9, we send the request with no additional parameters.

The sample call shown in the figure invokes a server-side program with URL "MyServlet", passing a request parameter named "myparm" with value set to 'abc'.

Sending a Request: Raw JavaScript, cont'd

```
1     function callback()
2     {
3       if (req.readyState == 4)
4       {
5         if (req.status == 200)
6         {
7           var text = req.responseText;
8           . . . // update the page
9         }
10        else
11        {
12          alert("Problem: " + req.statusText);
13        }
14      }
15    }
```

Here we show the JavaScript callback function that we registered to handle the response. It turns out that there are several possible ways that the browser invokes this function for a single request. Here we ignore all of the notifications except for the one that indicates that the request is complete (status = 200). At that point, we could then use the DOM to modify the page to contain the data received from the server, which is referenced by the "text" variable in line 7..

Note that this function uses the global "req" variable we showed on the last page.

Updating the Document: Raw JavaScript

- To update the HTML page, the **callback handler** can use the DOM to replace or update a **text node** containing the response text

```
function callback()
{
  . . .
  var text = req.responseText;
  setParagraph(text);
  . . .
}

function setParagraph(text)
{
  var myPara=document.getElementById('c');
  myPara.firstChild.nodeValue=text;
}
```

After the callback handler retrieves the response, it can use the DOM to change the page to display the returned data. Here we show the callback handler calling a JavaScript function that locates a paragraph by ID and then changes its text-node child to contain the text returned from the server.

What is JSON?

- JSON is the **JavaScript Object Notation** and was originally syntax in JavaScript for initializing objects, but has now become a standard data format rivaling XML

```
{
  "id"  : 17,
  "name": "Sue Smith",
  "gpa":3.45
}
```

You can read more about JSON at http://www.json.org.

Basic JSON Syntax

- JSON has a simple, easy to read and understand syntax, both for humans and computers

```
-- sample object
{
  "gpa"      : 3.33
  "name"     : "Sue Smith",
  "address"  :
  {
     "street":      "123 Elm",
     "city"  :      "Marion"
  }
}

-- sample array
[ "red", "green", "blue" ]
```

Objects can contain sub-objects and arrays. Arrays can contain objects and sub-arrays. And so forth.

Note that you must use double quotes rather than single quotes for values.

Using JSON in Raw JavaScript

- JavaScript applications can use JSON to create and initialize objects and arrays

```
var rect1 =
{
   upperLeft: { x: 2, y: 2 },
   lowerRight: { x: 4, y: 5 }
};

var str = JSON.stringify(rect1);

var rect2 = JSON.parse(str);
```

JavaScript also provides the eval() method that parses strings containing JSON, but this is potentially unsafe since it actually executes as JavaScript code. Also note that the JSON.stringify and JSON.parse methods require an ECMAScript version 5 implementation.

JSON is actually a subset of the JavaScript "object-literal notation" – in other words, every legal JSON string is also legal JavaScript OLN.

Popular JavaScript libraries such as jQuery also often include JSON support.

jQuery and Ajax

- jQuery provides rich support for invoking Ajax requests:
    - **load()** method to retrieve HTML from a server and apply it to a page element
    - **getJSON()** global method to retrieve server data as JSON
    - **get()** global method to retrieve arbitrary server content, including XML
    - Ajax progress-reporting callbacks
    - Ajax error-handling callback

We will not cover the load() method in this chapter. Note that there's also a low-level ajax() global method that gives you complete control over the Ajax request/response – we don't cover it in this chapter.

Processing JSON

- You can use the jQuery **getJSON()** global method to retrieve JSON content via Ajax
- The getJSON() method passes a native JavaScript object to your processing function so you can process the returned data

```
// server returns a JSON array:
//  ["BMW","Chevrolet","Honda","Mazda","Subaru","Toyota"]
$.getJSON("GetManufacturerListJSON", function(theList)
{
  for(var i = 0; i < theList.length; i++)
  {
     // do something
  }
});
```

The first parameter to getJSON() is the URL of the data; here we assume that it's a relative URL, but of course, it could be a fully-qualified URL beginning with "http://".

The second parameter here is a function that getJSON() will invoke when the Ajax request is complete, passing the function an object. Here we use an anonymous function that treats the object as an array, looping through each element.

Note that if you need to pass an HTTP parameter to the service, you can do so by passing another argument BEFORE the function argument. jQuery will append a question mark to the URL, followed by the argument.

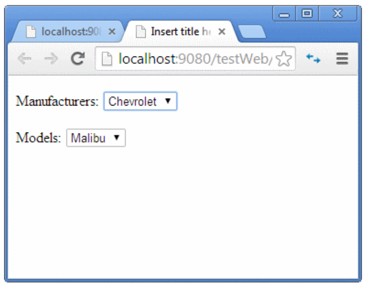
```
var param = "manufacturer=BMW";
$.getJSON("GetModelListJSON", param, function(theList)
{
 . . .
});
```

Complete JSON Example

test-jquery-ajax-json.html

GetManufacturerListJSON.java

GetModelListJSON.java



Manufacturers: Chevrolet

Models: Malibu

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<script src="jquery-1.11.1.js"></script>
<script>
   function populateModels(manuf)
   {
       var param = "manufacturer=" + manuf;

       $("#modelList").empty();

       $.getJSON("GetModelListJSON", param, function(theList)
       {
           for(var i = 0; i < theList.length; i++)
           {
             $("#modelList").append("<option>" +
                theList[i] + "</option>");
           }

       });
   }

  $(document).ready(function()
  {
    // get manufacturer list from server
    // in JSON
    $.getJSON("GetManufacturerListJSON", function(theList)
    {
       for(var i = 0; i < theList.length; i++)
       {
         $("#manufList").append("<option>" +
            theList[i] + "</option>");
       }

       // get first manufacturer, then populate model list,
       var first = theList[0];
       populateModels(first);
    });

    // set up handler to populate model list
    $("#manufList").change(function()
    {
       var manuf = $(this).val();
```

```
          populateModels(manuf);
      });
    });
  </script>
  </head>
  <body>
      <p>
        Manufacturers:
        <select id="manufList">
        </select>
      </p>
      <p>
        Models:
        <select id="modelList">
        </select>
      </p>
  </body>
  </html>
```

```java
package servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
@WebServlet("/GetManufacturerListJSON")
public class GetManufacturerListJSON extends HttpServlet
{
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();

        StringBuffer sb = new StringBuffer();
        sb.append(
          "[\"BMW\",\"Chevrolet\",\"Honda\",\"Mazda\",\"Subaru\",\"Toyota\"]");

        String retVal = sb.toString();
        out.println(retVal);
        System.out.println("In doGet: " + retVal);
    }
}
```

```java
package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.concurrent.TimeUnit;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
@WebServlet("/GetModelListJSON")
public class GetModelListJSON extends HttpServlet
{
    private HashMap<String, String[]> modelList =
            new HashMap<String, String[]>();

    private String[] bmwModels = {"328i", "535i", "755p"};
    private String[] chevyModels = {"Malibu", "Impala", "Volt"};
    private String[] hondaModels = {"Civic", "Accord", "Pilot"};
    private String[] mazdaModels = {"RX-8", "Miata" };
    private String[] subaruModels = {"Impreza", "Forester", "WRX"};
    private String[] toyotaModels = {"Corolla", "Highlander"};

    public GetModelListJSON()
    {
        modelList.put("BMW", bmwModels);
        modelList.put("Chevrolet", chevyModels);
        modelList.put("Honda", hondaModels);
        modelList.put("Mazda", mazdaModels);
        modelList.put("Subaru", subaruModels);
        modelList.put("Toyota", toyotaModels);
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("application/json");
        PrintWriter out = response.getWriter();

        String manufacturer = request.getParameter("manufacturer");
        System.out.println("Manufacturer: " + manufacturer);
        String[] list = modelList.get(manufacturer);

//          String s = null;
//          s.length();

        // simulate a lengthy process
        try
        {
            TimeUnit.SECONDS.sleep(5);
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        StringBuffer sb = new StringBuffer();
        sb.append("[");

        for(String model : list)
            sb.append("\"" + model + "\",");

        // remove last comma
        sb.deleteCharAt(sb.length() - 1);

        sb.append("]");

        String retVal = sb.toString();
        System.out.println("In doGet: " + retVal);
        out.print(retVal);
    }
}
```

jQuery and XML

- You can use the jQuery **get()** global method to retrieve XML content via Ajax
- You can then use JavaScript and jQuery DOM techniques to process the returned data

```
// server returns an XML document:
// <manufacturers>
//     <name>BMW</name>
//     <name>Chevrolet</name>
// </manufacturers>
$.get("GetManufacturerListXML", function(xml)
{
  var theList = $(xml).find("name");

  for(var i = 0; i < theList.length; i++)
  {
    // do something
  }
});
```

jQuery also provides a "post()" method that is similar, but executes an HTTP POST instead of a GET.

Note that if you need to pass HTTP parameters to the service, you can pass a JavaScript "hash-table" object:

```
var param = {manufacturer:  "BMW"};

$.get("GetModelListXML", param, function(xml)
{
 . . .
});
```

Complete XML Example

test-jquery-ajax-xml.html

GetManufacturerListXML.java

GetModelListXML.java

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
<script src="jquery-1.11.1.js"></script>
<script>
   function populateModels(manuf)
   {
       var param = {manufacturer: manuf};

       $("#modelList").empty();

       $.get("GetModelListXML", param, function(xml)
       {
           var theList = $(xml).find("model");

           for(var i = 0; i < theList.length; i++)
           {
             $("#modelList").append("<option>" +
                theList[i].firstChild.nodeValue + "</option>");
           }
       });
   }

  $(document).ready(function()
  {
    // get manufacturer list from server
    // in JSON
    $.get("GetManufacturerListXML", function(xml)
    {
       var theList = $(xml).find("name");

       for(var i = 0; i < theList.length; i++)
       {
         $("#manufList").append("<option>" +
            theList[i].firstChild.nodeValue + "</option>");
       }

       // get first manufacturer, then populate model list,
       var first = theList[0].firstChild.nodeValue;
       populateModels(first);
    });

    // set up handler to populate model list
    $("#manufList").change(function()
    {
       var manuf = $(this).val();
       populateModels(manuf);
    });
  });
</script>
</head>
<body>
    <p>
      Manufacturers:
      <select id="manufList">
      </select>
    </p>
    <p>
      Models:
      <select id="modelList">
      </select>
    </p>
</body>
</html>
```

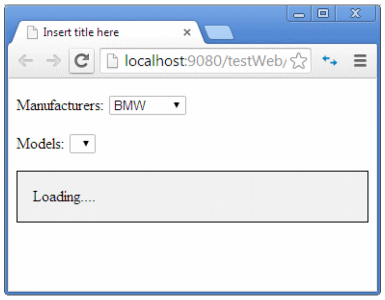
```java
package servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
@WebServlet("/GetManufacturerListXML")
public class GetManufacturerListXML extends HttpServlet
{
    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/xml");
        PrintWriter out = response.getWriter();

        StringBuffer sb = new StringBuffer();
        sb.append("<manufacturers>");
        sb.append("<name>BMW</name>");
        sb.append("<name>Chevrolet</name>");
        sb.append("<name>Honda</name>");
        sb.append("<name>Mazda</name>");
        sb.append("<name>Subaru</name>");
        sb.append("<name>Toyota</name>");
        sb.append("</manufacturers>");


        String retVal = sb.toString();
        out.println(retVal);
        System.out.println("In doGet: " + retVal);
    }
}
```

```java
package servlets;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@SuppressWarnings("serial")
@WebServlet("/GetModelListXML")
public class GetModelListXML extends HttpServlet
{
    private HashMap<String, String[]> modelList =
            new HashMap<String, String[]>();

    private String[] bmwModels = {"328i", "535i", "755p"};
    private String[] chevyModels = {"Malibu", "Impala", "Volt"};
    private String[] hondaModels = {"Civic", "Accord", "Pilot"};
    private String[] mazdaModels = {"RX-8", "Miata" };
    private String[] subaruModels = {"Impreza", "Forester", "WRX"};
    private String[] toyotaModels = {"Corolla", "Highlander"};

    public GetModelListXML()
    {
        modelList.put("BMW", bmwModels);
        modelList.put("Chevrolet", chevyModels);
        modelList.put("Honda", hondaModels);
        modelList.put("Mazda", mazdaModels);
        modelList.put("Subaru", subaruModels);
        modelList.put("Toyota", toyotaModels);
    }

    protected void doGet(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException
    {
        response.setContentType("text/xml");
        PrintWriter out = response.getWriter();

        String manufacturer = request.getParameter("manufacturer");
        System.out.println("Manufacturer: " + manufacturer);
        String[] list = modelList.get(manufacturer);

        StringBuffer sb = new StringBuffer();
        sb.append("<models>");

        for(String model : list)
            sb.append("<model>" + model + "</model>");

        sb.append("</models>");

        String retVal = sb.toString();
        System.out.println("In doGet: " + retVal);
        out.print(retVal);
    }
}
```

Monitoring Ajax Requests

- For lengthy Ajax operations, you can hook into the Ajax **callbacks** to give the user progress feedback

```
$(document).ajaxStart(function()
{
  $("#progress").show();
});

$(document).ajaxStop(function()
{
  $("#progress").hide();
});
. . .
<div id="progress>
  Loading...
</div>
```



Manufacturers: BMW

Models:

Loading....

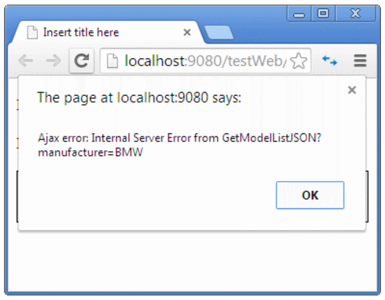
The code shown here is the document.ready() method BEFORE any Ajax invocations.

In early versions of jQuery, you could attach these callbacks to any page element, but now the documentation recommends that you only attach them to the "document" object.

Handling Errors

- You can register an Ajax error-handling function so you can report or log errors
- The error-handling function optionally accepts parameters that describe the error

```
$(document).ajaxError(
  function(event, jqXHR, ajaxSettings, thrownError)
{
    alert("Ajax error: " + thrownError +
          " from " + ajaxSettings.url);
});
```



Insert title here

localhost:9080/testWeb/

The page at localhost:9080 says:

Ajax error: Internal Server Error from GetModelListJSON?manufacturer=BMW

OK

In early versions of jQuery, you could attach this callback to any page element, but now the documentation recommends that you only attach them to the "document" object.

Chapter Summary

In this chapter, you learned:

- The fundamentals of Ajax in JavaScript
- How to work with Ajax in jQuery