Lesson 1: Browser Detection
   Introduction
      The Navigator Object

Lesson 2: An Introduction to AJAX
   Understand AJAX

Lesson 3: XML
   An Introduction to XML
      Creating Your First XML Document
         XML Attributes

Lesson 4: AJAX: Your First Application
   The XML Document
      The HTML Document

Lesson 5: AJAX: Phonebook
   The XML Document
      The HTML Document

Lesson 6: String Objects
   String Objects: The match method
   String Objects: The substr method

Lesson 7: AJAX: Retrieving Information from a Perl Script
   The Perl Script
      The HTML Document
      Editing the Perl Script

Lesson 8: AJAX: Sending Information to a Perl Script
   The Perl Script: Read
      The Perl Script: Write
      The HTML Document

Lesson 9: Form Validation
   Required Fields
      Valid Value: First Name
         
   Valid Value: Email Address

Lesson 10: Google Maps and the API
   Using Google Maps

Lesson 11: Cookies
An Introduction to Cookies
Making It More Complex

Lesson 12: **AJAX: Making your Application User Friendly**

- The Disadvantages of AJAX
- Bookmarks and the Back Button
- Updating the User

*Copyright © 1998-2014 O'Reilly Media, Inc.*

*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.*
Browser Detection

Introduction

Welcome to class! In this course you will be learning advanced JavaScript. Throughout the course you will be working to create a complex phonebook application that will use form validation, cookies, and most importantly, AJAX, a relatively new type of remote scripting. Remote scripting AJAX is handled differently based on the browser being used to view the page, so before you can learn any AJAX, you must first learn about browser detection.

Let's get started!

Note

All of the examples, quizzes, and objectives for this course should be completed using the HTML editor in the Sandbox. If you have never taken an O'Reilly course that utilizes this editor, please click here to learn about the different features of this tool.

The Navigator Object

Now that your scripts are becoming more complex, they might not give the same results on all browsers. Because of this, it may be necessary to determine which browser is being used to view your page, and then deliver the appropriate code. You can do this using the Navigator object.

The Navigator object contains all information about the browser being used to display the webpage. The information with which you will likely be most interested is the browser name and the browser version.

In HTML, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">
<!--
function doIt(){
    var bName=navigator.appName
    var bVersion=navigator.appVersion
    alert("browser name: "+bName)
    alert("browser version: "+bVersion)
}
//-->
</script>
</head>
<body>
<form>
    <input type="button" onclick="doIt()" value="Do It" />
</form>
</body>
</html>
```

Click Preview.

Note

Keep in mind that every time you Preview a file, your changes will be saved. Think about whether you want the previous code overwritten or not. If not, use File Save As before you Preview.
When you click the button you should see two alerts. The first alert will show you the name of the browser being used to view the page, while the second alert shows the version.

**Note**

If you're using Firefox or Safari to view your page, the browser name returned is **Netscape**. If you're using Internet Explorer to view your page, the browser name returned is **Microsoft Internet Explorer**. This is because the code for some sites is written such that it will only allow these two browsers. If you need to know the browser more specifically, you might find the following references helpful:

- Internet Explorer, Firefox, or Opera
- Safari or Konqueror

Suppose you have a website with elements whose position depends on the width of the window displaying it. Referencing this property will be done differently depending on which browser you are using. Let's see this in action.

**In HTML, type the following into the Editor below:**

```html
<html>
<head>
<script type="text/javascript">
<!--
function doIt(){
    if(navigator.appName == "Netscape"){
        if (window.innerWidth > 500){
            document.getElementById("someText").style.left = 300
        } else{
            document.getElementById("someText").style.left = 200
        }
    }
    if(navigator.appName == "Microsoft Internet Explorer"){
        if (document.body.clientWidth > 500){
            document.getElementById("someText").style.left = 300
        } else{
            document.getElementById("someText").style.left = 200
        }
    }
}
//-->
</script>
</head>
<body>
<span id="someText" style="position:absolute; left:120;">Here is a some text</span>
<br/>
<form>
    <input type="button" onclick="doIt()" value="Do It" />
</form>
</body>
</html>
```
Click Preview. When the button is selected the object someText will have a new position based on the width of the window.

Fortunately, browsers are being made to adhere to web standards more and more closely now. As a result, delivering separate code based on each browser being used has become less important. You will see, however, that it is still important in the exciting area of AJAX programming. You will learn about this more as you continue through the course. See you in the next lab!
An Introduction to AJAX

AJAX, which stands for *Asynchronous JavaScript and XML*, is a relatively new type of remote scripting that can be used to create webpages that are faster and more dynamic. In the past, any data exchanged between the user and the server required that the entire webpage be loaded and displayed. Using AJAX, small quantities of data can be exchanged between the user and the server, and changes can be displayed without having to reload the webpage. The result are pages that are more dynamic and user friendly.

As you go through this lesson, keep in mind that AJAX is not a new programming language. It is simply a way to use JavaScript and XML to deliver content and create webpages that are faster and more responsive.

**Understand AJAX**

AJAX allows information to be delivered to a webpage without having to reload the page. This information is stored within documents on the server. Users can interact with the server the entire time they are using the webpage, and information flows smoothly between the server and the webpage.

To experience this for yourself, check out the following links:

**Traditional Webpage**

**AJAX Webpage**

Notice that the entire page is refreshed upon clicking the *Search Phonebook* button on the traditional webpage. This is not the case for the webpage that uses AJAX. Take a moment to view the source code for both of these pages and compare the codes. What differences do you notice?

As you continue to learn more about AJAX, you might find it helpful to keep analogies like the one discussed in this lesson in mind. In the next lesson you'll learn how to create your own XML documents from which you can deliver information. See you there!

*Copyright © 1998-2014 O'Reilly Media, Inc.*

*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.*
An Introduction to XML

XML stands for eXtensible Markup Language. Data is stored in XML documents. XML is actually an agreement among people (programming people) to store and share textual data using the same method.

For example, suppose I presented you with the following information:

John Smith
52
Arkansas

Given only this information, you couldn't possibly know what it's supposed to be used for or what it means. XML gives us a way to mark up this data so that it's understandable by people and other computer programs. For example, the data above might be marked up like this:

<PLAYERNAME>John Smith</PLAYERNAME>
<PLAYERNUMBER>52</PLAYERNUMBER>
<PLAYERUNIVERSITY>Arkansas</PLAYERUNIVERSITY>

Now you know a lot more about the meaning of this information. We know that this is likely some kind of sports player, his number is 52, and he plays for the University of Arkansas. The markup above isn't like HTML. It's not telling us or a program how the content should look. It's telling us the meaning of the content. Later we'll see that we can use style sheets to convert our content into a display.

Creating Your First XML Document

For your first XML document, we'll construct a very basic file for storing phonebook information. The type of information we want to store in this XML file is quite simple:

- First Name
- Last Name
- Phone Number
- Address

How can we implement this in XML? First we have to define a set of tags we want to use to keep track of our information. These tags are similar to the tags in HTML because both HTML and XML are derived from SGML. Unlike HTML though, XML has no predefined tags. Tags in XML are those you create or that have been created by other developers. XML tags are also referred to as elements.

The following elements will be used in this example:

- <FIRST>
- <LAST>
- <PHONE>
- <ADDRESS>
- <PHONEBOOK>
- <LISTING>
Let's get started with XML! Switch to XML by clicking on the Syntax menu and changing to XML.

Let's make an XML file. This XML file will markup information that would go into a phonebook. Type everything exactly as you see it in the box below:

```
<?xml version="1.0"?>
<!DOCTYPE PHONEBOOK>
<PHONEBOOK>
  <LISTING>
    <FIRST>John</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-123-4567</PHONE>
  </LISTING>
</PHONEBOOK>
```

So what have we created? This XML file should be really easy to understand. Even if you didn't know anything about XML, you would probably guess that this file contains a PHONEBOOK, and that phonebook contains one LISTING for John Smith, 1-800-123-4567.

Let's go over exactly what these lines mean. The first line of code reading `<?xml version="1.0"?>` indicates that the file we are creating is an XML file. The second line, `<!DOCTYPE PHONEBOOK>` specifies that we are creating the document type PHONEBOOK. You will learn more about DOCTYPE later, but for now remember that your DOCTYPE must match the first set of tags in your XML document. In this case, the !DOCTYPE is PHONEBOOK. Notice that `<PHONEBOOK>` is the first tag of our document. The other lines in the XML document define your phonebook and data. In this example, we are tracking FIRST name, LAST name, and PHONE number. How would you add another listing?

Before we go any further with our practice, let me tell you few more details about XML:

- When entering XML, remember XML is **NOT** HTML! HTML tags are not automatically defined in XML.
- XML is case sensitive. `<stuff>` and `<STUFF>` are **NOT** the same thing!
- All XML tags must be closed. This means for every `<STUFF>` a matching `</STUFF>` must exist. `<STUFF/>` is an example of an single tag that is closed.
- All XML tags must be properly nested. This means that tags should be closed in the opposite order from which they are opened.
- `<?xml version="1.0"?>` must be the first thing in your XML file. No whitespace can be present before this text.

Let's check that we've typed the xml document correctly. When you're using XML syntax, you should see **Check Syntax**, **Validate** and **Translate** icons in the toolbar. Click on **Check Syntax** now.

If your document is written correctly, you will see No Errors Found at the bottom of the pop-up window.

After you get this XML document working, let's add another listing. Add the green content below to your XML document:
In XML, type the following into the editor below:

```xml
<?xml version="1.0"?>
<!DOCTYPE PHONEBOOK>
<PHONEBOOK>
  <LISTING>
    <FIRST>John</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-123-4567</PHONE>
    <ADDRESS>302 E. John St, Champaign IL 61820</ADDRESS>
  </LISTING>
  <LISTING>
    <FIRST>Jane</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-234-5678</PHONE>
    <ADDRESS>1005 Gravenstein Highway N, Sebastopol CA 95472</ADDRESS>
  </LISTING>
</PHONEBOOK>
```

Be sure to Check Syntax to ensure that your document is working. If it is, Save your file as phone.xml as this file will be used in future lessons and projects.

**XML Attributes**

Let's change the XML document a bit. Suppose John Smith has two phone numbers - one for work and one for his cell phone. How can we add both numbers to your phone book in a way that makes sense?

One solution may be this:

```xml
<?xml version="1.0"?>
<!DOCTYPE PHONEBOOK>
<PHONEBOOK>
  <LISTING>
    <FIRST>John</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-123-4567</PHONE>
    <PHONE>1-555-222-3333</PHONE>
  </LISTING>
  <LISTING>
    <FIRST>Jane</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-234-5678</PHONE>
  </LISTING>
</PHONEBOOK>
```

Although we now have two phone numbers listed for him, it's impossible to determine which is his work and which is his cell phone number.

In order to address issues like this, sometimes data needs to be stored using something other than simple elements. Using a modifier will help us perform this task. XML has features that function as "modifiers," but they are called Attributes. They don't perform their magic without elements, however.

Let's say we are working with our XML based phonebook above. We realize that many of our friends have multiple phone numbers. We could create new elements for each of these phones, such as <HOME_PHONE>, <CELL_PHONE>, etc. This would make it clear to anybody just what data is specified by the tags, but there is a more elegant solution to the problem.

**ATTRIBUTES!** Instead of making a unique tag for the each type of phone number (home, cell, or work), let's specify the phone type within the PHONE tag itself.
In XML, type the following into the editor below:

```xml
<?xml version="1.0"?>
<!DOCTYPE PHONEBOOK>
<PHONEBOOK>
  <LISTING>
    <FIRST>John</FIRST>
    <LAST>Smith</LAST>
    <PHONE TYPE="CELL">1-800-123-4567</PHONE>
    <PHONE TYPE="WORK">1-555-222-3333</PHONE>
    <ADDRESS>302 E. John St, Champaign IL 61820</ADDRESS>
  </LISTING>
  <LISTING>
    <FIRST>Jane</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-234-5678</PHONE>
    <ADDRESS>1005 Gravenstein Highway N, Sebastopol CA 95472</ADDRESS>
  </LISTING>
</PHONEBOOK>
```

Now we know exactly what each phone number indicates! **Check Syntax** to make sure your XML document works. Then be sure to **Save**.

Now that you’re familiar with XML, you’ve taken the first step in learning about AJAX. Feel free to play around and experiment with XML to build your confidence. See you in the next lesson!

*Copyright © 1998-2014 O'Reilly Media, Inc.*

*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.*
Congratulations! You now know enough about HTML, CSS, JavaScript, and XML to create your first AJAX application. This application won’t be complex, but it will teach you the basics so you can create more sophisticated applications in future lessons.

The XML Document

The first thing you’ll need in order to create your application is the XML document. This document will store the information that will be delivered to your HTML document.

```
<?xml version="1.0"?>
<!DOCTYPE GREETING>
<GREETING>
  <HELLO>hello world!</HELLO>
</GREETING>
```

Check Syntax on this document to be sure that it works. When you’re certain that it does, save the file as ajaxgreeting.xml.

The HTML Document

Now let’s create the HTML document that will be used to retrieve information from the XML document.
In HTML, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if(browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxgreeting.xml', true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    if(http.readyState == 4) {
        var response = http.responseXML.documentElement;
        var helloobj = response.getElementsByTagName("HELLO")
        alert(helloobj[0].firstChild.data)
    }
}

</script>
</head>

<body>
<form>
    <input type="button" value="SendGreeting" onClick="sndReq()" />
</form>
</body>
</html>
```

Click Preview. You should see an html file with a button on the page. When you select the button, an alert will open that reads hello world! Let's take a closer look at how this works:

When the html file is opened in a browser window, the `createRequestObject` function is called. This function detects the browser being used to view the file so that an HTTP request object can be created. This object allows you to change a page with data from the server after the page has loaded. The syntax to create this object differs slightly depending on the browser being used. In any case, the request object is stored in a variable called `ro`, which is the value returned by the function. Since the function is "called" using:

```
var http = createRequestObject()
```

the request object returned is stored in `http`.

Upon clicking the button on the page, the function `sndReq` is called and three lines of code are executed. The `open` method is used to open the `ajaxgreeting.xml` from which we want to get data. The syntax used for the `open` method is:

```
requestObject.open("method", "URL", async)
```
**Method** can be set to either **get** or **post**. You use **get** to retrieve information from the server, and **post** to send information to it.

**URL** should be the file from which you wish to retrieve or send information.

**Async** is used to specify whether data is handled asynchronously. (This value will almost always be set to **true**. The only reason to set it to **false** would be if you were retrieving information from a very bulky document.)

After the open method is called, the **onreadystatechange** property is set so that the **handleResponse** function will be called any time the state of the request object changes. You can probably see why an **if** statement is needed in the **handleResponse**. If this statement was not there, the function would be called multiple times instead of only when we want it to be called. Check it out for yourself. Try replacing the **handleResponse** function with the following code where the **if** statement has been removed:

```
function handleResponse() {
    alert("hello!")
}
```

What happens?

After specifying the function that will be called, the **send** method is used to contact the server. Since information is being retrieved from the server, there is no need to send any data to it and we use **NULL**. If you were sending information to the server, you would place the data to be sent within the parentheses.

When the **handleResponse** function is called and the state of the request object is **equal to 4**, three lines of code are executed. These are the lines used to obtain specific data from the XML document!

**responseXML** is a property of the request object. It is used to return the response from the server as XML, and it returns an XML document object. This means that all of the methods and properties of an XML document object are now available to you! In this case, the **response** array is set equal to the **documentElement** property. This property represents the root node of the document. The root node of **ajaxgreeting.xml** is **<GREETING>** since it is the first element in the XML document.

Once this property is stored in the **response** array, you can access the other elements or data stored within **<GREETING>**. In this case, **<GREETING>** only contains the element **<HELLO>**. The **helloobj** array is then set equal to the **documentElement** property of the **response** array. This allows us to access any elements or data stored within **<HELLO>**.

The final line is a line with which you should be very familiar by now. It is used to alert the **data** stored in the **first** **<HELLO>** tag found in the XML document.

You have now created your first AJAX application! Be sure you understand the basics about how the script works since it will become more complex as you continue through the course.

*Copyright © 1998-2014 O'Reilly Media, Inc.*

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.
Now that you've learned a little about AJAX, it's time to move on to creating a more complex (and more useful!) application. For the next few lessons we'll be working our way up to creating a phonebook.

**The XML Document**

The first thing you need in order to create your phonebook is the XML document. This document stores the information that will be delivered to your HTML document.

In XML, type the following into the Editor below:

```xml
<?xml version="1.0"?>
<!DOCTYPE PHONEBOOK>
<PHONEBOOK>
  <LISTING>
    <FIRST>John</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-123-4567</PHONE>
    <ADDRESS>302 E. John St, Champaign IL 61820</ADDRESS>
    <EMAIL>john@smith.com</EMAIL>
  </LISTING>
  <LISTING>
    <FIRST>Jane</FIRST>
    <LAST>Smith</LAST>
    <PHONE>1-800-234-5678</PHONE>
    <ADDRESS>1005 Gravenstein Highway N, Sebastopol CA 95472</ADDRESS>
    <EMAIL>jane@smith.com</EMAIL>
  </LISTING>
</PHONEBOOK>
```

Check Syntax on this document to be sure that it works. When you are certain that it does, save the file as ajaxphone.xml.

**The HTML Document**

Now let's create the HTML document that will be used to retrieve information from the XML document.
In HTML, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
      ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
      ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxphone.xml', true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    if (http.readyState == 4) {
        var response = http.responseXML.documentElement
        listings = response.getElementsByTagName("LISTING")
        for (i = 0; i < listings.length; i++) {
            nameobj = listings[i].getElementsByTagName("FIRST")
            if (nameobj[0].firstChild.data == document.getElementById("first").value) {
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                document.getElementById("address").innerHTML = addressobj[0].firstChild.data
                document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
            }
        }
    }
}<script type="text/javascript">

<form id="search">
    <input type="text" id="first" />
    <input type="button" value="Search Phonebook" onClick="sndReq()" />
</form>

<div id="address"></div>
<div id="phone"></div>
</form>
</body>
</html>
```
Click **Preview**. When the page opens, type either **John** or **Jane** into the text field, then click on the **Search Phonebook** button. An address or phone number should appear on the page, just below the text field.

Let's take a closer look at the code used to do this. Much of the script should look familiar to you from the previous lesson. In fact, the only function that is different is the **handleResponse** function.

When the **handleResponse** function is called and the state of the request object is **equal to 4**, the **response** array is set equal to the **ajaxphone.xml** document object. Remember, the **documentElement** property represents the root node of the document, which in this case is **<PHONEBOOK>**, the first element in the XML document.

Next, the **listings** array is set equal to the data and elements stored within the **LISTING** tags. A **for** loop is used to take a closer look at the information within each **<LISTING>** so that only the information for which we are searching will be returned.

Each time the loop is executed, all data stored within the **FIRST** tag will be stored in the **nameobj** array. We can then compare this **data** with the **value** entered into the text field.

If these values are equal, the **addressobj** and **phoneobj** are set equal to the data stored within the **PHONE** and **ADDRESS** tags for that particular **listing**. The **innerHTML** of the **address** and **phone div** tag is then set equal to this **data**. Note that two empty **div** tags had to be created with HTML in order to do this.

As you continue through the course, you'll be building on this AJAX application to make a more complex phonebook. Be sure to save the above files before moving on to the next lesson!

*Copyright © 1998-2014 O'Reilly Media, Inc.*

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.
Welcome to lesson 6! Let's continue working with the phonebook application. The goal here is to add the ability to search your listings by matching only part of the name for which you are searching. For example, if you were to type J into the text field, all first names starting with this letter would be returned.

In order to make this happen, you will use the JavaScript String object. This object allows you to examine and manipulate strings. Let's get started!

**String Objects: The match method**

The match method can be used to match a string within a string. Let's take a look at this using our phonebook HTML file.
In HTML, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">

function createRequestObject() {
var ro
  var browser = navigator.appName
  if (browser == "Microsoft Internet Explorer") {
    ro = new ActiveXObject("Microsoft.XMLHTTP")
  } else {
    ro = new XMLHttpRequest()
  }
  return ro
}

var http = createRequestObject()

function sndReq() {
  http.open('get', 'ajaxphone.xml', true)
  http.onreadystatechange = handleResponse
  http.send(null)
}

function handleResponse() {

  var found = 0
  if (http.readyState == 4) {
    if (http.responseText.length > 0) {
      var response = http.responseXML.documentElement
      listings = response.getElementsByTagName("LISTING")
      for (i = 0; i < listings.length; i++) {
        firstobj = listings[i].getElementsByTagName("FIRST")
        if (firstobj[0].firstChild.data == document.getElementById("first").value) {
          firstobj = listings[i].getElementsByTagName("FIRST")
          lastobj = listings[i].getElementsByTagName("LAST")
          addressobj = listings[i].getElementsByTagName("ADDRESS")
          phoneobj = listings[i].getElementsByTagName("PHONE")
          emailobj = listings[i].getElementsByTagName("EMAIL")

          document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " " + lastobj[0].firstChild.data
          document.getElementById("address").innerHTML = addressobj[0].firstChild.data
          document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
          document.getElementById("email").innerHTML = emailobj[0].firstChild.data

          found = 1
        } else if (firstobj[0].firstChild.data.match(document.getElementById("first").value)) {
          firstobj = listings[i].getElementsByTagName("FIRST")
        }
      }
    }
  }
}
```

Click Preview. When the page opens, type J into the text field, then click on the Search Phonebook button. You should see the names John and Jane listed just below the text field. This is because both John and Jane contain the letter J.

The match method uses the syntax:

```javascript
stringObj.match(pattern)
```

In this case the data stored within the FIRST tags is tested to see if it contains a string equal to the value entered into the text field. If a match is found, the name appears on the document.

Now try typing the letter x into the text field. What happens? You should see an alert stating There were no matches.

What happens if you type the letter n into the text field? You should see the names John and Jane listed just below the text field. This is the expected result since both John and Jane contain the letter n. However, it is probably not the desired result. It would probably be better is to search using the first letter only.

**String Objects: The substr method**
<html>
<head>
<script type="text/javascript">
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxphone.xml', true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    var found = 0
    if(http.readyState == 4){
        document.getElementById("theName").innerHTML = ""
        document.getElementById("address").innerHTML = ""
        document.getElementById("phone").innerHTML = ""
        document.getElementById("email").innerHTML = ""

        var searchLength = document.getElementById("first").value.length
        var response = http.responseXML.documentElement
        listings = response.getElementsByTagName("LISTING")

        for (i=0;i<listings.length;i++) {
            firstobj = listings[i].getElementsByTagName("FIRST")
            if (firstobj[0].firstChild.data == document.getElementById("first").value){
                firstobj = listings[i].getElementsByTagName("FIRST")
                lastobj = listings[i].getElementsByTagName("LAST")
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                emailobj = listings[i].getElementsByTagName("EMAIL")

                document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " " + lastobj[0].firstChild.data
                document.getElementById("address").innerHTML = addressobj[0].firstChild.data
                document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
                document.getElementById("email").innerHTML = emailobj[0].firstChild.data
            }else if (firstobj[0].firstChild.data.substr(0, searchLength)
Click Preview. When the page opens type J into the text field. You should see the same results as you did in the previous example. Now type the n into the text field. This time no matches were found because no first names beginning with the letter n are entered into phonebook.xml.

The substr method was used to do this. The substr method uses the syntax:

```
stringObj.substr(start, length)
```

In order to search the data stored within the FIRST tags from the first letter, we must set start equal to 0. The length of the string should equal the length of the string stored in document.search.first.value.

Using this method, you can search using the first letter. However you can also search using any portion of the beginning of the string. To see this in action, try typing "Jo" into the text field. What happens?

Our phonebook application is really starting to take shape! In the next lesson, you'll learn how to use Perl scripts to refine it even more. See you there!

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
So far you have used JavaScript to retrieve information from an XML document. In this lesson you will learn how to retrieve information from a Perl script. Though you may not be familiar with the intricacies of Perl scripting, it's important to see how this process works. The reason for this will become clearer in the next few lessons.

**The Perl Script**

Perl is a scripting language that can be used for a variety of tasks, including system administration, web development, and network programming. Although it may appear to be far more complex than the XML documents with which you worked previously, Perl has an advantage in that you will be able to read from and write information to the script.

Let's take a closer look.

```perl
#!/usr/bin/perl
print "Content-type: text/html\n\n";
$query = split(/=/, $ENV{'QUERY_STRING'});
$searchname = $query{'firstname'};

%addressdb = ('John' => '302 E. John St, Champaign IL 61820',
              'Jane' => '1005 Gravenstein Highway N, Sebastopol CA 95472');

%phonedb = ('John' => '1-800-123-4567',
            'Jane' => '1-800-234-5678');

$address = $addressdb{'$searchname'};
$phone = $phonedb{'$searchname'};
print "$address|$phone";
```

Click Check Syntax. Name the file **ajaxphone.pl**. CodeRunner will save and then debug your Perl script. After a few seconds you should see a Results window with the file name and the syntax OK message. This means the debugging was successful.

**Note** Be sure that the line `#!/usr/bin/perl` is the very first line of your Perl script. There cannot be any blank lines or spaces preceding it!

**The HTML Document**

Now let's create the HTML document that will be used to retrieve information from the Perl script.
In HTML mode, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
        ro = new XMLHttpRequest()
    }
    return ro
}
var http = createRequestObject()
function sndReq(action) {
    http.open('get', 'cgi/ajaxphone.pl?firstname=\'+action, true)
    http.onreadystatechange = handleResponse
    http.send(null)
}
function handleResponse() {
    if (http.readyState == 4) {
        var response = http.responseText;
        var update = new Array()
        if (response.indexOf('|')) {
            update = response.split('|')
            document.getElementById('theName').innerHTML = document.getElementById('first').value
            document.getElementById('address').innerHTML = update[0]
            document.getElementById('phone').innerHTML = update[1]
        }
    }
}
</script>
</head>
<body>
<form id="search">
<input type="text" id="first" />
<input type="button" value="Search Phonebook" onClick="sndReq(document.getElementById('first').value)" />
</form>
<div id="theName"></div>
<div id="address"></div>
<div id="phone"></div>
</body>
</html>
```

Click Preview. When the page opens, type either John or Jane into the text field, then click on the Search Phonebook button. The results should look very familiar to you! However, there are some slight differences in how these results were achieved.
Take a look how the `sndReq` function is called. Notice that the value entered into the first text field is passed to the function and stored in the `action` variable. This makes it so that this value can be passed to and used by the Perl script.

The `handleResponse` function has also changed. When this function is called and the state of the request object is equal to 4, the `response` variable is set equal to the text returned by the server. In this case, the text returned originated from the last statement in our Perl script:

```
print "\$address|\$phone";
```

The `update` array is created by splitting this text so that `\$address` will be the first element in the array, and `\$phone` will be the second element in the array. We can then set the `innerHTML` of our div tags equal to these elements.

**Editing the Perl Script**

Suppose you wanted to change your application so that the email address belonging to the person for whom you are searching is also displayed. Even if you do not know anything about Perl, you can add the necessary information to do this.

In PERL mode, type the following into the Editor below:

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";
$\text{query} = \text{split}(/=/, $\text{ENV} \{ \text{QUERY\_STRING} \});
$\text{searchname} = $\text{query} \{ \"\text{firstname}\" \};

$\text{addressdb} = (\text{\'John\'} => \text{\'302 E. John St, Champaign IL 61820\'},
                 \text{\'Jane\'} => \text{\'1005 Gravenstein Highway N, Sebastopol CA 95472\'});

$\text{phonedb} = (\text{\'John\'} => \text{\'1-800-123-4567\'},
                 \text{\'Jane\'} => \text{\'1-800-234-5678\'});

$\text{emaildb} = (\text{\'John\'} => \text{\'john@domain.com\'},
                 \text{\'Jane\'} => \text{\'jane@domain.com\'});

$\text{address} = $\text{addressdb} \{ \text{\$searchname}\};
$\text{phone} = $\text{phonedb} \{ \text{\$searchname}\};
$\text{email} = $\text{emaildb} \{ \text{\$searchname}\};

print "$\text{address}|\$phone|\$email";
```

Click [Debug Perl](#). Name the file `ajaxphone.pl`. Be sure that your debugging was successful. If it wasn't, email your mentor.

Take a look at the new lines added to the script. First, we associated `John` and `Jane` to their respective email addresses, using a `=>` to separate them. We then copied the syntax used in the lines:

```
$\text{address} = $\text{addressdb} \{ \text{\$searchname}\};
$\text{phone} = $\text{phonedb} \{ \text{\$searchname}\};
$\text{email} = $\text{emaildb} \{ \text{\$searchname}\};
```

Then we added a `|` and `value` to the end of the text to be printed by the Perl script. Remember, this is the text returned to the HTML document.

Now let's edit the HTML document so that we can use the additional information returned to it.
In HTML mode, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript">
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if(browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq(action) {
    http.open('get', 'cgi/ajaxphone.pl?firstname='+action, true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    if(http.readyState == 4){
        var response = http.responseText;
        var update = new Array()
        if(response.indexOf('|')) {
            update = response.split('|')
            document.getElementById('theName').innerHTML = document.getElementById('first').value
            document.getElementById('address').innerHTML = update[0]
            document.getElementById('phone').innerHTML = update[1]
            document.getElementById('email').innerHTML = update[2]
        }
    }
}
</script>
</head>
<body>
<form id="search">
    <input type="text" id="first" />
    <input type="button" value="Search Phonebook" onClick="sndReq(document.getElementById('first').value)"
</form>
<div id="theName"></div>
<div id="address"></div>
<div id="phone"></div>
<div id="email"></div>
</body>
</html>

Click Preview. This time when you search for John you should see the email address returned as well. In
order to accomplish this we simply added a `div` tag in which to display the information. We then set the `innerHTML` of the `div` tag equal to the third element in the `update` array.

In the next lesson you'll learn how to add listings to your phonebook application using a Perl script. Be sure you understand the concepts discussed here before moving on to the next lesson.
AJAX: Sending Information to a Perl Script

In the last lesson you learned how to retrieve information from a Perl script to your webpage. The result was similar to the one achieved when retrieving information from an xml file. The advantage of using a Perl script, however, is that you can also write new information to the file.

Note
For this lesson save your Perl files in the cgi folder. That folder has write permissions set and a pre-loaded Perl library file, which you'll need. Be sure to update the html file to reflect the correct path to your Perl scripts as well.

The Perl Script: Read

Check it out.

In PERL, type the following into the Editor below:

```perl
#!/usr/bin/perl

print "Content-type: text/html\n\n";
%

$query = split(/=/, $ENV{QUERY_STRING});

$searchname = $query["firstname"];

open(DATAFILE, "ajaxphone.txt") or die "Cannot open file: \"ajaxphone.txt\" \n";  
while($line = <DATAFILE>) {
  ($firstname,$address,$phone,$email) = split(/\|\/, $line);
  $addressdb["$firstname"] = $address;
  $phonedb["$firstname"] = $phone;
  $emaildb["$firstname"] = $email;
}

close(DATAFILE);

$address = $addressdb["$searchname"];
$phone = $phonedb["$searchname"];
$email = $emaildb["$searchname"];

print "$address|$phone|$email";
```

Click Preview. Name the file **ajaxphoneread.pl**. After a few seconds you should see the syntax OK message.

This script should look very similar to the one used in the previous lesson. The only difference is that now we're retrieving data from the text file called **ajaxphone.txt** instead of from within the script itself. This file will be created and information will be written to it using a second script.

The Perl Script: Write
In PERL, type the following into the Editor below:

```perl
#!/usr/bin/perl

print "Content-type: text/html\n\n";

require "ajaxlib.pl";

parse_input;

# Check if ajaxphone.txt file exists. If not, create it.
if (! ( -f "ajaxphone.txt")) {
    open (DATAFILE,">ajaxphone.txt" );
    close (DATAFILE);
}

open (DATAFILE, ">>ajaxphone.txt" );
flock (DATAFILE, 2) || die "Could not lock the data file\n";
pเสมอ FORM_DATA{‘entryName’}|$FORM_DATA{‘entryAddress’}|
prเสมอ FORM_DATA{‘entryPhone’}|$FORM_DATA{‘entryEmail’}\n"
flock (DATAFILE, 8);
close (DATAFILE);
```

Click **Preview**. Name the file **ajaxphonewrite.pl**. After a few seconds you should see the **syntax OK** message.

Don't let this script overwhelm you! The only line you would ever need to change is the print statement. For example, if you wanted to add an image for each entry in the text field, you would use:

```perl
print DATAFILE "$FORM_DATA{‘entryName’}|$FORM_DATA{‘entryAddress’}|
prเสมอ FORM_DATA{‘entryPhone’}|$FORM_DATA{‘entryEmail’}|$FORM_DATA{‘imgSrc’}\n";
```

Note that the text file will be created by this script if it does not already exist.

**The HTML Document**

Now let's create the HTML document that will be used to both write information to and retrieve information from the Perl script.
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if(browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq(action) {
    http.open('get', 'cgi/ajaxphoneread.pl?firstname='+action, true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function sndWrite() {
    var entryName=document.getElementById("entryName").value
    var entryAddress=document.getElementById("entryAddress").value
    var entryPhone=document.getElementById("entryPhone").value
    var entryEmail=document.getElementById("entryEmail").value

    entryName=escape(entryName)
    entryAddress=escape(entryAddress)
    entryPhone=escape(entryPhone)
    entryEmail=escape(entryEmail)

    var firstString='entryName='+entryName
    var addressString='entryAddress='+entryAddress
    var phoneString='entryPhone='+entryPhone
    var emailString='entryEmail='+entryEmail

    var sendString=firstString+'&'+addressString+'&'+phoneString+'&'+emailString

    http.open('post', 'cgi/ajaxphonewrite.pl');
    http.send(sendString);
}

function handleResponse() {
    if(http.readyState == 4){
        var response = http.responseText;
        var update = new Array()
        if(response.indexOf('|')) {
            update = response.split('|')
            document.getElementById('theName').innerHTML = document.getElementById('first').value
            document.getElementById('address').innerHTML = update[0]
            document.getElementById('phone').innerHTML = update[1]
            document.getElementById('email').innerHTML = update[2]
        }
    }
}
Click Preview. The Search Phonebook button works exactly the same way as it did in the previous lesson.

Now look at the top of the page. Here you will see a new form. Fill in the text fields to create a new entry, then select the button labeled Send Entry. When you are done, search for your new entry using the first name you just entered. You should see the new information returned.

**Note** If you are having trouble entering information or retrieving information once it is entered, please email your mentor at this time.

When the Send Entry button is clicked, the sndWrite function is called. Each value entered into the text field
is stored as a variable, then the variables are escaped. The escape function is used to encode a string so that it can be read on all computers.

Next, string variables are assigned using the format:

```javascript
stringVariable='identifier='+entryValue
```

So, for example, if the name entered into the entryName text field is Tom, then it would look like this:

```javascript
firstString='entryName=Tom'
```

Finally ampersands & are placed between each of these variables and the entire string is stored in sendString. There, finally a string the script can recognize! It is imperative that the information sent to the script follows this format.

The final two lines of the function are used to send sendString to ajaxphonewrite.pl. Note that we are using post here instead of get as you used in the past since we are sending information instead of receiving it.

Congratulations! Your phonebook is definitely starting to become a more exciting application. However, there are still some improvements to be made. In the next lesson you will learn how to check your entries to ensure that the information stored in the text field is valid. See you there!

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
Form Validation

In the last lesson you learned how to store information entered into your HTML document using a Perl script. But it's not enough to simply store this information. You must also make sure that the information stored is valid. JavaScript can be used to check the validity of the information before it is sent to the script.

Required Fields

Any new entry added to the phonebook must contain a first name. It is the one piece of information that is required to identify each entry.

Let's take a closer look.
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq(action) {
    http.open('get', 'cgi/ajaxphoneread.pl?firstname=' + action, true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function sndWrite() {
    var entryName=document.getElementById("entryName").value
    var entryAddress=document.getElementById("entryAddress").value
    var entryPhone=document.getElementById("entryPhone").value
    var entryEmail=document.getElementById("entryEmail").value
    if (entryName == "") {
        alert("Please enter a First Name.")
    } else {
        entryName=escape(entryName)
        entryAddress=escape(entryAddress)
        entryPhone=escape(entryPhone)
        entryEmail=escape(entryEmail)
        var firstString='entryName='+entryName
        var addressString='entryAddress='+entryAddress
        var phoneString='entryPhone='+entryPhone
        var emailString='entryEmail='+entryEmail
        var sendString=firstString+'&'+addressString+'&'+phoneString+'&'+emailString
        http.open('post', 'cgi/ajaxphonewrite.pl');
        http.send(sendString);
    }
}

function handleResponse() {
    if(http.readyState == 4) {
        var response = http.responseText;
        var update = new Array()
        if(response.indexOf('|')) {
            update = response.split('|')
            document.getElementById('theName').innerHTML = document.getElementById('first')
        }
    }
}
Click Preview. When the page opens, click the Send Entry button. You should be alerted to Please enter a First Name.

Take a look at the code added to the sndWrite function. An if statement is used to check whether or not the variable is empty "". If it is empty, an alert is displayed. If it is not empty, the information is sent to the Perl script where it will be stored in a text file.
Valid Value: First Name

Let's take the above example one step further. Suppose we want to make sure that the first name entered contains only capital or lowercase letters. To do this you would have to utilize the `match` method.
<html>
<head>
<script type="text/javascript">

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq(action) {
    http.open('get', 'cgi/ajaxphoneread.pl?firstname='+action, true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function sndWrite() {
    var entryName=document.getElementById("entryName").value
    var entryAddress=document.getElementById("entryAddress").value
    var entryPhone=document.getElementById("entryPhone").value
    var entryEmail=document.getElementById("entryEmail").value

    if (entryName.length == 0 || entryName.match(/[^a-zA-Z]/)) {
        alert("Please enter a First Name. Be sure that the name contains only letters.")
    } else {
        entryName=escape(entryName)
        entryAddress=escape(entryAddress)
        entryPhone=escape(entryPhone)
        entryEmail=escape(entryEmail)

        var firstString='entryName='+entryName
        var addressString='entryAddress='+entryAddress
        var phoneString='entryPhone='+entryPhone
        var emailString='entryEmail='+entryEmail

        var sendString=firstString+'&'+addressString+'&'+phoneString+'&'+emailString

        http.open('post', 'cgi/ajaxphonewrite.pl');
        http.send(sendString);
    }
}

function handleResponse() {
    if (http.readyState == 4) {
        var response = http.responseText;
        var update = new Array()

        if (response.indexOf('|')) {
            update = response.split('|')
        }
    }
</script>
</head>
</html>
Click Preview. When the page opens, enter a number in the text field labeled First Name. Click the Send Entry button. You should be alerted to Please enter a First Name. Be sure that the name contains only letters.

To do this we used the match method. Rememeber what we learned lesson 6? Match method uses the syntax:
stringObj.match(pattern)

In this case, we used:

```
entryName.match(/^[^a-zA-Z]/)
```

The `entryName` is checked to ensure that there are only letters. Note that we are not matching an exact string here. Instead we are matching a pattern. Because of this, we must enclose the pattern within `forward slashes`.

**Square brackets** are used to search the string for a range of values. For example:

```
[A-Z]
```

matches any capital letter, A through Z. To match either a capital or lowercase letter, you would use:

```
[A-Za-z]
```

In this case, we wish to match any characters that are *not* capital or lowercase letters, so we must include a *carat*. A carat literally means **not**.

```
[^A-Za-z]
```

Here are some examples of other values you can use:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[aeiouAEIOU]</td>
<td>matches any lower OR upper case vowel</td>
</tr>
<tr>
<td>[^aeiouAEIOU]</td>
<td>matches a non-vowel character. The ^ means NOT when inside a bracket</td>
</tr>
<tr>
<td>[0-9]</td>
<td>matches any single digit</td>
</tr>
<tr>
<td>[^0-9]</td>
<td>matches any character that is not a digit.</td>
</tr>
<tr>
<td>[a-z]</td>
<td>matches any lower case letter</td>
</tr>
<tr>
<td>[a-zA-Z]</td>
<td>matches any lower OR upper case letter</td>
</tr>
<tr>
<td>[a-zA-Z0-9]</td>
<td>matches any digit or letter</td>
</tr>
</tbody>
</table>

* You can match any character by using a period.

```
.n..e
```

This will match the letters *n* and *e* with any two characters between them. It will therefore match `name, nine, nZWe, n3be`, and *n/(e*.

* You can use an asterisk (*) to find zero or more occurrences of a character.

```
[nN]e*d
```

This will match *Ned, need, nd*, and *neeeed*.

* You can use a question mark (?) to find zero or one occurrence of a character.

```
[nN]e?d
```

This will match *Ned, nd*, and *ned*.

**Valid Value: Email Address**

Every email address contains an @ character. The @ character is never the first character of an email address. We can check to ensure that this is the case using the `indexOf` method.
<html>
<head></head>
<body>
  <script type="text/javascript">
    function createRequestObject() {
      var ro
      var browser = navigator.appName
      if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
      } else {
        ro = new XMLHttpRequest()
      }
      return ro
    }

    var http = createRequestObject()

    function sndReq(action) {
      http.open('get', 'cgi/ajaxphoneread.pl?firstname=' + action, true)
      http.onreadystatechange = handleResponse
      http.send(null)
    }

    function sndWrite() {
      var entryName = document.getElementById("entryName").value
      var entryAddress = document.getElementById("entryAddress").value
      var entryPhone = document.getElementById("entryPhone").value
      var entryEmail = document.getElementById("entryEmail").value

      if (entryName == "" || entryName.match(/[^a-zA-Z]/)) {
        alert("Please enter a First Name. Be sure that the name contains only letters."
      }
      else if (entryEmail.indexOf("@") == -1 || entryEmail.indexOf("@") == 0) {
        alert("Please be sure the email address uses the format name@domain.com")
      }
      else {
        entryName = escape(entryName)
        entryAddress = escape(entryAddress)
        entryPhone = escape(entryPhone)
        entryEmail = escape(entryEmail)

        var firstString = 'entryName=' + entryName
        var addressString = 'entryAddress=' + entryAddress
        var phoneString = 'entryPhone=' + entryPhone
        var emailString = 'entryEmail=' + entryEmail

        var sendString = firstString + '&' + addressString + '&' + phoneString + '&' + emailString

        http.open('post', 'cgi/ajaxphonewrite.pl');
        http.send(sendString);
      }
    }

    function handleResponse() {
      if (http.readyState == 4) {
        var response = http.responseText;
        var update = new Array()
      }
  
  </script>
</body>
</html>
Click **Preview**. When the page opens, enter information for a new entry. When you go to enter the **Email** address, do
not include the @ character. Then click the Send Entry button. You should be alerted to Please be sure the email address uses the format name@domain.com.

The indexOf method returns the position of the first occurrence of a specified value. Here we use an if statement to ensure that the entryEmail contains an @ character. If it does not, entryEmail.indexOf("@") returns a -1. We also want to ensure that the @ is not the first character of the email address, so we check to see that entryEmail.indexOf("@") does not return 0.

Our phonebook is really coming along! In the next lesson we will continue to make it more complex using Google maps. See you there!
Until now, our phonebook application has transferred only small pieces of data, such as text and images. We used AJAX so that we could update this information without refreshing the page, but because the data served on our page was not very server-intensive, we did not really save much in terms of efficiency. Well no more! In this lesson, you'll begin to appreciate AJAX.

If you have ever used Google Maps before, then you may have found yourself impressed with their ease of use. See this again for yourself! Take a look at Google Maps. Be sure to scroll around the map and zoom in on a couple of locations.

Let's add some Google Maps to our phonebook so that we can see where our friends live. To do this, we'll be using the Google Maps API. An API is an Application Programming Interface. You can think of the API as another library of functions that our phonebook application can use.

**Using Google Maps**

Let's take a look at what you can do. Check out the demos. Be sure to read the Terms of Use. Also check the Developer's Guide and Tutorial links, and read through the Geocoding section.

Let's write some code to generate maps for the addresses stored in ajaxphone.xml.
<html>
<head>
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
</head>
<script type="text/javascript">
var geocoder;
var map;
var marker;

function load() {
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(37.4419, -122.1419);
    var myOptions = {
        zoom: 13,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    map = new google.maps.Map(document.getElementById("mymap"), myOptions);
}

function showAddress(theAddress) {
    var myaddress = theAddress
    if (geocoder) {
        geocoder.geocode( { 'address': myaddress}, function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                map.setCenter(results[0].geometry.location);
                var marker = new google.maps.Marker({
                    map: map,
                    position: results[0].geometry.location
                });
            } else {
                alert("Geocode was not successful for the following reason: " + status);
            }
        });
    }
}

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxphone.xml', true)
http.onreadystatechange = handleResponse
http.send(null)
}

function handleResponse() {
    if(http.readyState == 4){
        document.getElementById("theName").innerHTML = ""
        document.getElementById("address").innerHTML = ""
        document.getElementById("phone").innerHTML = ""
        document.getElementById("email").innerHTML = ""

        var response = http.responseXML.documentElement
        listings=response.getElementsByTagName("LISTING")

        for (i=0;i<listings.length;i++) {
            firstobj = listings[i].getElementsByTagName("FIRST")
            if (firstobj[0].firstChild.data == document.getElementById("first").value){
                firstobj = listings[i].getElementsByTagName("FIRST")
                lastobj = listings[i].getElementsByTagName("LAST")
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                emailobj = listings[i].getElementsByTagName("EMAIL")

                document.getElementById("theName").innerHTML = firstobj[0].firstChild.data
                document.getElementById("address").innerHTML = addressobj[0].firstChild.data
                document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
                document.getElementById("email").innerHTML = emailobj[0].firstChild.data
                theAddress = addressobj[0].firstChild.data
                showAddress(theAddress)
            }
        }
    }
}
</script>
</head>

<body onload="load()">
<form id="search">
    <input type="text" id="first" />
    <input type="button" value="Search Phonebook" onClick="sndReq()" />
</form>
<div id="theName"></div>
<div id="address"></div>
<div id="phone"></div>
<div id="email"></div>
<div id="mymap" style="width: 500px; height: 300px"></div>
</body>
</html>

Click Preview. Be sure to type either John or Jane in the text field, then click the button labeled Search Phonebook. You should see the map change to reveal the address associated with the name you typed. Let's take a closer look at the code.

The first thing added was a set of opening and closing script tags. These allow us to specify the location of the Google Maps JavaScript file to be used with our script. Note that the URL includes a parameter to indicate whether this application uses a sensor to determine the user's location. We've set this parameter to false.
Next, let's create the variables called map, geocoder and marker. The map variable will be used in the load function, which is called in the onload of the webpage. The other two variables, geocoder and marker, are used in the showAddress function.

The load function is used to create a Map options object. This object contains the information used to create the map: initial location, zoom level, type of map. We set latlang to the latitude and longitude of Palo Alto, California, US. center uses latlang to center the initial map at that point. Feel free to center your map anywhere you’d like.

We set the zoom level on our map to 13 and the initial type to ROADMAP. Other types include SATELLITE, HYBRID and TERRAIN. Play with those as well.

The load function is also used to create a new geocoder. This allows for direct communication with the Google servers so that we can see maps of our addresses.

The showAddress function is exactly the same as the code you saw in the Google Maps API documentation. It is called from the handleResponse function, and theAddress associated with the person for whom we are searching is passed to the function upon calling it.

geometry is a method of geocoder and is used to translate theAddress into latitude and longitude. Once these are obtained, setCenter is used to re-center the map and a marker is placed on this point on the map.

The geocoding example you experimented with above is only one of the many things that the Google Maps API allows you to do. Take some time and use the documentation and references mentioned above to check out what else is available. See you in the next lesson!

Copyright © 1998-2014 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.
An Introduction to Cookies

If you have ever registered at a website, chances are you are familiar with cookies. A cookie is a file used to store information about users. You can also use cookies to tailor that information to be delivered to individual specifications. The cookie file is stored locally, on your browser, and it is maintained between browser sessions. It will remain there even if the browser is closed or if you completely shut down your computer.

Suppose the xml file that stores the data from your phonebook has grown considerably. The last search you did was for an important work contact, but now you can't seem to remember the name of that person. To find it, you could search through the entire XML file in the hopes that one of the names will jog your memory, or you could use a cookie to recall the name that has been searched most recently. Let's take a closer look.
In HTML, type the following into the Editor below:

```html
<html>
<head>
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false">
</script>

<script type="text/javascript">
var recent1 = initCookie("recent1")

function initCookie(cookieName){
    if (document.cookie.indexOf(cookieName + "=") == -1){
        return ""
    }
    else {
        return getCookie(cookieName)
    }
}

function getCookie(cookieName){
    start=document.cookie.indexOf(cookieName + "=") + cookieName.length + 1
    end=document.cookie.indexOf(";", start)
    if (end == -1){
        end = document.cookie.length
    }
    cookieLength=end-start
    return(document.cookie.substr(start,cookieLength))
}

function setRecent(mostRecentName){
    recent1=mostRecentName
    document.cookie="recent1="+recent1
}

var geocoder;
var map;
var marker;
function load() {
    alert(document.cookie)
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(37.4419, -122.1419);
    var myOptions = {
        zoom: 13,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    map = new google.maps.Map(document.getElementById("mymap"), myOptions);
}
```

function showAddress(theAddress) {
    var myaddress = theAddress

    if (geocoder) {
        geocoder.geocode({ 'address': myaddress }, function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                map.setCenter(results[0].geometry.location);

                var marker = new google.maps.Marker({
                    map: map,
                    position: results[0].geometry.location
                });
            } else {
                alert("Geocode was not successful for the following reason: " + status);
            }
        });
    }
}

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if (browser == "Microsoft Internet Explorer") {
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    } else {
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxphone.xml', true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    if (http.readyState == 4) {
        document.getElementById("theName").innerHTML = ""
        document.getElementById("address").innerHTML = ""
        document.getElementById("phone").innerHTML = ""
        document.getElementById("email").innerHTML = ""

        var response = http.responseXML.documentElement
        listings = response.getElementsByTagName("LISTING")

        for (i = 0; i < listings.length; i++) {
            firstobj = listings[i].getElementsByTagName("FIRST")
            if (firstobj[0].firstChild.data == document.getElementById("first").value) {
                firstobj = listings[i].getElementsByTagName("FIRST")
                lastobj = listings[i].getElementsByTagName("LAST")
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                emailobj = listings[i].getElementsByTagName("EMAIL")

                document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " "+ lastobj[0].firstChild.data
document.getElementById("address").innerHTML = addressobj[0].firstChild.data
document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
document.getElementById("email").innerHTML = emailobj[0].firstChild.data

theAddress = addressobj[0].firstChild.data
showAddress(theAddress)

alert(document.cookie)

mostRecentName = firstobj[0].firstChild.data
setRecent(mostRecentName)

</script>
</head>
<body onload="load()"
<form id="search"
   <input type="text" id="first" />
   <input type="button" value="Search Phonebook" onClick="sndReq()" />
</form>
<div id="theName"></div>
<div id="address"></div>
<div id="phone"></div>
<div id="email"></div>
<div id="mymap" style="width: 500px; height: 300px"></div>
</body>
</html>

Click **Preview**. When the page opens, type John or Jane into the text field and click the **Search Phonebook** button. You might have to do this a couple of times in order to alert text. You should see something like recent1=John or recent1=Jane

Take some time and add a few entries to your ajaxphone.xml file if you have not done so already. This will give you additional names with which to test your cookies.

**Note** If you are having trouble getting this to work, be sure you have cookies enabled in your browser settings, then clear your current cookies. Also, if you are using Internet Explorer you should make sure that your domain is added to the list of trusted sites. If you are still having trouble, save your file and try viewing it using a “regular” browser window.

A cookie can be set and retrieved using `document.cookie`. Although you can simply set `document.cookie` equal to a value, like John, it is best to identify this value with a label, such as recent1=John. There are several steps that must occur in order to do this.

When the page opens, the cookie is initialized using the `initCookie` function. This function checks to see if the string recent1= is stored as a cookie. If it does not exist, the value "" is returned from the function and stored in the variable recent1. If it exists, the `getCookie` function is called.

The `getCookie` function is used to obtain the string following recent1= that is stored as a cookie. In order to do this, you must set `start` equal to the position of the = in recent1=. Because multiple cookies are separated by a semi-colon, an attempt is made to set `end` equal to the first ; following start. If this semi-colon does not exist, end is instead set equal to the final character position in the string. By subtracting start from end, the length of the string can be determined, thereby allowing you to search for and return the substring.

Each time the Search Phonebook button is selected, `document.cookie` is displayed in an alert box. Then the name that is currently being searched is sent to the setRecent function.

The setRecent function assigns `document.cookie` equal to "recent1="+recent1 If, for example, the value of recent1 is John, the cookie would be set to recent1=John.
Making It More Complex

Being able to recall the last name for which you searched is nice, but it would be even nicer to recall the last two names, and to list these names directly on the webpage instead of using an alert. Let's give this a try!
<html>
<head>
</head>
<body>
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
var recent1 = initCookie("recent1")
var recent2 = initCookie("recent2")
function initCookie(cookieName){
    if (document.cookie.indexOf(cookieName + "=") == -1){
        return ""
    } else {
        return getCookie(cookieName)
    }
}
function getCookie(cookieName){
    start=document.cookie.indexOf(cookieName + "=") + cookieName.length + 1
    end=document.cookie.indexOf(";", start)
    if (end == -1) {
        end = document.cookie.length
    }
    cookieLength=end-start
    return(document.cookie.substr(start,cookieLength))
}
function setRecent(mostRecentName){
    recent2=recent1
    recent1=mostRecentName
    document.cookie="recent1"=recent1
    if (recent2.length >= 1) {
        document.cookie="recent2"=recent2
    }
}
function getRecent(){
    if (recent2){
        document.getElementById("R1").innerHTML=getCookie("recent1")
        document.getElementById("R2").innerHTML=getCookie("recent2")
    } else{
        document.getElementById("R1").innerHTML=getCookie("recent1")
    }
}
</script>
</body>
</html>
function load() {

  getRecent()
  geocoder = new google.maps.Geocoder();
  var latlng = new google.maps.LatLng(37.4419, -122.1419);
  var myOptions = {
    zoom: 13,
    center: latlng,
    mapTypeId: google.maps.MapTypeId.ROADMAP
  }
  map = new google.maps.Map(document.getElementById("mymap"), myOptions);
}

function showAddress(theAddress) {
  var myaddress = theAddress

  if (geocoder) {
    geocoder.geocode({ 'address': myaddress}, function(results, status) {
      if (status == google.maps.GeocoderStatus.OK) {
        map.setCenter(results[0].geometry.location);
        var marker = new google.maps.Marker({
          map: map,
          position: results[0].geometry.location
        });

      } else {
        alert("Geocode was not successful for the following reason: " + status);
      }
    });
  }
}

function createRequestObject() {
  var ro
  var browser = navigator.appName
  if(browser == "Microsoft Internet Explorer"){
    ro = new ActiveXObject("Microsoft.XMLHTTP")
  }else{
    ro = new XMLHttpRequest()
  }
  return ro
}

var http = createRequestObject()

function sndReq() {
  http.open('get', 'ajaxphone.xml', true)
  http.onreadystatechange = handleResponse
  http.send(null)
}

function handleResponse() {
  if(http.readyState == 4){
    document.getElementById("theName").innerHTML = ""
    document.getElementById("address").innerHTML = ""
    document.getElementById("phone").innerHTML = ""
    document.getElementById("email").innerHTML = ""
  }
```html
var response = http.responseXML.documentElement
listings=response.getElementsByTagName("LISTING")
for (i=0;i<listings.length;i++) {
  firstobj = listings[i].getElementsByTagName("FIRST")
  if (firstobj[0].firstChild.data == document.getElementById("first").value){
    firstobj = listings[i].getElementsByTagName("FIRST")
    lastobj = listings[i].getElementsByTagName("LAST")
    addressobj = listings[i].getElementsByTagName("ADDRESS")
    phoneobj = listings[i].getElementsByTagName("PHONE")
    emailobj = listings[i].getElementsByTagName("EMAIL")
    document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " " + lastobj[0].firstChild.data
    document.getElementById("address").innerHTML = addressobj[0].firstChild.data
    document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
    document.getElementById("email").innerHTML = emailobj[0].firstChild.data
    theAddress = addressobj[0].firstChild.data
    showAddress(theAddress)
  }
}
</script>
</head>

<body onload="load()">
<form id="search">
  <input type="text" id="first" />
  <input type="button" value="Search Phonebook" onClick="sndReq()" />
</form>
<div id="theName"></div>
<div id="address"></div>
<div id="phone"></div>
<div id="email"></div>
<div id="mymap" style="width: 500px; height: 300px"></div>
<div id="recent">
  Most Recent Searches
  <ul>
    <li id="R1"></li>
    <li id="R2"></li>
  </ul>
</div>
</body>
</html>

Click Preview. This time you should see the two most recently searched for names in a list, just after the map.

Note: If you are having trouble getting this to work, be sure you have cookies enabled in your browser settings, then clear your current cookies. Also, if you are using Internet Explorer you should make sure that your domain is added to the list of trusted sites. If you are still having trouble, save your file and try viewing it using a "regular" browser window.
In order to display two names, you must assign values to two cookies. The `initCookie` function must therefore be called twice. Notice that the second time it is called, `recent2` is passed to the function.

The `setRecent` function is used to assign the value previously stored in `recent1` into `recent2` and to store this value as a second `cookie`.

Finally, the `getRecent` function was created to display the values stored in the `recent1` and `recent2` cookies. You do this using the `innerHTML` property of the `list items` created as HTML elements in our document. Notice that the `getRecent` function is called each time `Search Phonebook` is selected.

Cookies provide you, as a developer, with an opportunity to personalize the pages you make for your individual users. That is, they can help you to make your application more user friendly. In the next lesson you will learn about more tools that you can use to give your users the best possible experience.
The Disadvantages of AJAX

While AJAX is certainly an exciting new technology, as with any technology, there are certain disadvantages. In this lesson you’ll learn about two major disadvantages in using AJAX, and the tools that can be used to work around them.

Bookmarks and the Back Button

Open one of your AJAX projects in a browser window. Perform several searches, then select the Back button. Notice that even though the information on the webpage has changed several times, the back button will not allow you to see these changes. Let’s add some code to “fix” the back button.

Note

The following code will only work for the back button if you view the page using Firefox. The code required to make this work on Internet Explorer is rather complex. Please visit this site for more information.
<html>
<head>

<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></script>
</head>
<body>

var geocoder;
var map;
var marker;

function load() {
    geocoder = new google.maps.Geocoder();
    var latlng = new google.maps.LatLng(37.4419, -122.1419);
    var myOptions = {
        zoom: 13,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    map = new google.maps.Map(document.getElementById("mymap"), myOptions);
    checkHash()
    checkHash();
}

function showAddress(theAddress) {
    var myaddress = theAddress
    if (geocoder) {
        geocoder.geocode({ 'address': myaddress}, function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
                map.setCenter(results[0].geometry.location);
                var marker = new google.maps.Marker({
                    map: map,
                    position: results[0].geometry.location
                });
            } else {
                alert("Geocode was not successful for the following reason: " + status);
            }
        });
    } else {
        alert("Geocode was not successful for the following reason: " + status);
    }
}

function createRequestObject() {
    var ro
    var browser = navigator.appName
    if(browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {

</body>
</html>
http.open('get', 'ajaxphone.xml', true)
http.onreadystatechange = handleResponse
http.send(null)
}

function handleResponse() {
    if(http.readyState == 4){
        document.getElementById("theName").innerHTML = ""
        document.getElementById("address").innerHTML = ""
        document.getElementById("phone").innerHTML = ""
        document.getElementById("email").innerHTML = ""

        var response = http.responseXML.documentElement
        listings=response.getElementsByTagName("LISTING")

        for (i=0;i<listings.length;i++) {
            firstobj = listings[i].getElementsByTagName("FIRST")
            if (firstobj[0].firstChild.data == document.getElementById("first").value){
                firstobj = listings[i].getElementsByTagName("FIRST")
                lastobj = listings[i].getElementsByTagName("LAST")
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                emailobj = listings[i].getElementsByTagName("EMAIL")

                document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " " + lastobj[0].firstChild.data
                document.getElementById("address").innerHTML = addressobj[0].firstChild.data
                document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
                document.getElementById("email").innerHTML = emailobj[0].firstChild.data

                theAddress = addressobj[0].firstChild.data
                showAddress(theAddress)

                updateHistory(firstobj[0].firstChild.data)
            }
        }
    }
}

function checkHash () {
    if (window.location.hash) {
        var hashtext = window.location.hash.substring(1)
        if (hashtext != document.getElementById("histHelper").value) {
            document.getElementById("first").value = hashtext
            document.getElementById("histHelper").value = hashtext
            sndReq()
        }
    }
}

function updateHistory(newHash) {
    window.location.hash=newHash;
    document.getElementById("histHelper").value = newHash
}

</script>
</head>
<body onload="load()">
<form id="search"
Save this file, then open the page in Firefox. Try searching a few times for John, Jane, or any other names you may have put in ajaxphone.xml. Then click the Back button. You should be able to cycle through the information returned for all names you used.

A hidden text field called histHelper is added to the webpage. This text field will store the name of the person for whom information is currently displayed.

The load function is used to call the checkHash function upon loading the webpage. Additionally, the setInterval method is used to call this function every 1000 milliseconds, or every second.

The checkHash function checks to see if there is a value stored in window.location.hash. If there is, the hashtext variable is set equal to any string following the hash # character. If this string differs from the name for which information is currently being displayed, then the value of both the first and histHelper fields is set equal to the string. The sndReq function is called, and the information is updated to reflect that of the previous name for which you searched. In essence, then, checkHash is used to determine whether the back button has been clicked.

Notice that the updateHistory function is called from handleResponse. This function will add a new hash to the end of the URL in the form of #Name. Check out the address bar in the browser to see this in action.

The addition of #Name to the URL solves another problem. It allows you to bookmark a particular state of your webpage so that you can pass it along to friends or come back to it later. This works on both Firefox and Internet Explorer.

**Updating the User**

When you submit or retrieve information using AJAX techniques, you do so without reloading the webpage. This means that there are no visual cues to indicate that information is being transferred, such as waving icons or rotating circles. Regular internet users have come to expect and rely on these cues, so it’s a good idea to add code to your page to create other indicators.
function createRequestObject() {
    var ro
    var browser = navigator.appName
    if(browser == "Microsoft Internet Explorer"){
        ro = new ActiveXObject("Microsoft.XMLHTTP")
    }else{
        ro = new XMLHttpRequest()
    }
    return ro
}

var http = createRequestObject()

function sndReq() {
    http.open('get', 'ajaxphone.xml', true)
    http.onreadystatechange = handleResponse
    http.send(null)
}

function handleResponse() {
    if(http.readyState == 4){
        document.getElementById("theStatus").innerHTML=""
        document.getElementById("theName").innerHTML = ""
        document.getElementById("address").innerHTML = ""
        document.getElementById("phone").innerHTML = ""
        document.getElementById("email").innerHTML = ""

        var response = http.responseXML.documentElement
        listings=response.getElementsByTagName("LISTING")

        for (i=0;i<listings.length;i++) {
            firstobj = listings[i].getElementsByTagName("FIRST")
            if (firstobj[0].firstChild.data == document.getElementById("first").value){
                firstobj = listings[i].getElementsByTagName("FIRST")
                lastobj = listings[i].getElementsByTagName("LAST")
                addressobj = listings[i].getElementsByTagName("ADDRESS")
                phoneobj = listings[i].getElementsByTagName("PHONE")
                emailobj = listings[i].getElementsByTagName("EMAIL")

                document.getElementById("theName").innerHTML = firstobj[0].firstChild.data + " " + lastobj[0].firstChild.data
                document.getElementById("address").innerHTML = addressobj[0].firstChild.data
                document.getElementById("phone").innerHTML = phoneobj[0].firstChild.data
                document.getElementById("email").innerHTML = emailobj[0].firstChild.data
            }
        }
    }
}

function dotIt(){
Click **Preview**. Type either **John** or **Jane** in the text field and click the **Search Phonebook** button. Before displaying the information returned from the phonebook, you should see the text **Transferring Data...** displayed on the webpage.

When the **Search Phonebook** button is clicked, the **updateStatus** function is called. This function assigns the **innerHTML** of the **theStatus** div element equal to **Transferring Data**, and sets the **innerHTML** of all other elements so that nothing is displayed.

A **while** loop is used to call the **dotIt** function every 500 milliseconds, or every half of a second. Since the initial value of **j** is 500, and the value of **j** increases by 500 each time through the loop, this means the **dotIt** function is called four times, and four . are added to **Transferring Data**.

At the conclusion of the while loop, the **sndReq** function is called, and information from **ajaxphone.xml** is retrieved.

For your final project in this course you will be asked to put together the information learned throughout the course into a single phonebook application. As you design it and any future applications, think about how to give your users the most enjoyable experience possible. Good luck!

*Copyright © 1998-2014 O'Reilly Media, Inc.*

*This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See [http://creativecommons.org/licenses/by-sa/3.0/legalcode](http://creativecommons.org/licenses/by-sa/3.0/legalcode) for more information.*