15

Ajax-Enabled Rich Internet Applications

… the challenges are for the designers of these applications: to forget what we think we know about the limitations of the Web, and begin to imagine a wider, richer range of possibilities. It's going to be fun.

—Jesse James Garrett

Dojo is the standard library JavaScript never had.

—Alex Russell

To know how to suggest is the great art of teaching. To attain it we must be able to guess what will interest …

—Henri-Fredreic Amiel

It is characteristic of the epistemological tradition to present us with partial scenarios and then to demand whole or categorical answers as it were.

—Avrum Stroll

O! call back yesterday, bid time return.

—William Shakespeare

OBJECTIVES

In this chapter you will learn:

- What Ajax is and why it is important for building Rich Internet Applications.
- What asynchronous requests are and how they help give web applications the feel of desktop applications.
- What the `XMLHttpRequest` object is and how it's used to create and manage asynchronous requests to servers and to receive asynchronous responses from servers.

OBJECTIVES

- Methods and properties of the `XMLHttpRequest` object.
- How to use XHTML, JavaScript, CSS, XML, JSON and the DOM in Ajax applications.
- How to use Ajax frameworks and toolkits, specifically Dojo, to conveniently create robust Ajax-enabled Rich Internet Applications.
- About resources for studying Ajax-related issues such as security, performance, debugging, the “back-button problem” and more.

Outline

15.1 Introduction

- Usability of web applications has lagged behind compared to desktop applications
- Rich Internet Applications (RIAs)
  - Web applications that approximate the look, feel and usability of desktop applications
  - Two key attributes—performance and rich GUI
- RIA performance
  - Comes from Ajax (Asynchronous JavaScript and XML), which uses client-side scripting to make web applications more responsive
- Ajax applications separate client-side user interaction and server communication, and run them in parallel, making the delays of server-side processing more transparent to the user
- "Raw" Ajax uses JavaScript to send asynchronous requests to the server, then updates the page using the DOM
- When writing "raw" Ajax you need to deal directly with cross-browser portability issues, making it impractical for developing large-scale applications

15.1 Introduction (Cont.)

- **Portability issues**
  - Hidden by Ajax toolkits, such as Dojo, Prototype and Script.aculo.us
  - Toolkits provide powerful ready-to-use controls and functions that enrich web applications and simplify JavaScript coding by making it cross-browser compatible
- **Achieve rich GUI in RIAs with**
  - Ajax toolkits
  - RIA environments such as Adobe's Flex, Microsoft's Silverlight and JavaServer Faces
  - Such toolkits and environments provide powerful ready-to-use controls and functions that enrich web applications.
- **Client-side of Ajax applications**
  - Written in XHTML and CSS
  - Uses JavaScript to add functionality to the user interface
- **XML and JSON are used to structure the data passed between the server and the client**
- **`XMLHttpRequest`**
  - The Ajax component that manages interaction with the server
  - Commonly abbreviated as XHR.

15.2 Traditional Web Applications vs. Ajax Applications

- **Traditional web applications**
  - **User fills in the form's fields, then submits the form**
  - **Browser generates a request to the server, which receives the request and processes it**
  - **Server generates and sends a response containing the exact page that the browser will render**
  - **Browser loads the new page and temporarily makes the browser window blank**
  - **Client *waits* for the server to respond and *reloads the entire page* with the data from the response**
- **While a synchronous request is being processed on the server, the user cannot interact with the client web browser**
- **The synchronous model was originally designed for a web of hypertext documents**
  - **some people called it the "brochure web"**
  - **model yielded "choppy" application performance**

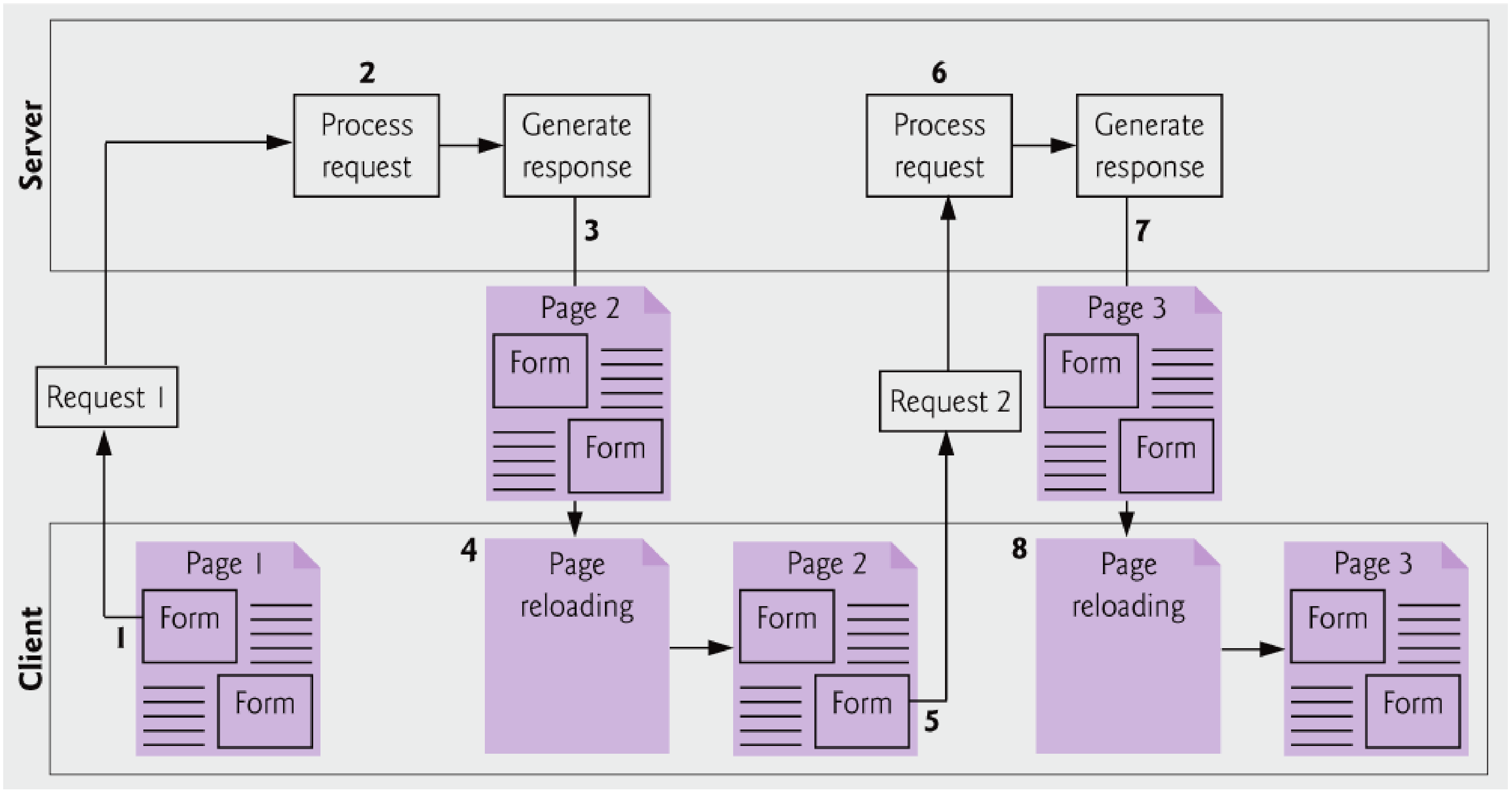


Fig. 15.1 | Classic web application reloading the page for every user interaction.

15.2 Traditional Web Applications vs. Ajax Applications (Cont.)

- **In an Ajax application, when the user interacts with a page**
  - **Client creates an `XMLHttpRequest` object to manage a request**
  - **`XMLHttpRequest` object sends the request to and awaits the response from the server**
  - **Requests are asynchronous, allowing the user to continue interacting with the application while the server processes the request concurrently**
  - **When the server responds, the `XMLHttpRequest` object that issued the request invokes a callback function, which typically uses partial page updates to display the returned data in the existing web page *without reloading the entire page***
- **Callback function updates only a designated part of the page**
- **Partial page updates help make web applications more responsive, making them feel more like desktop applications**

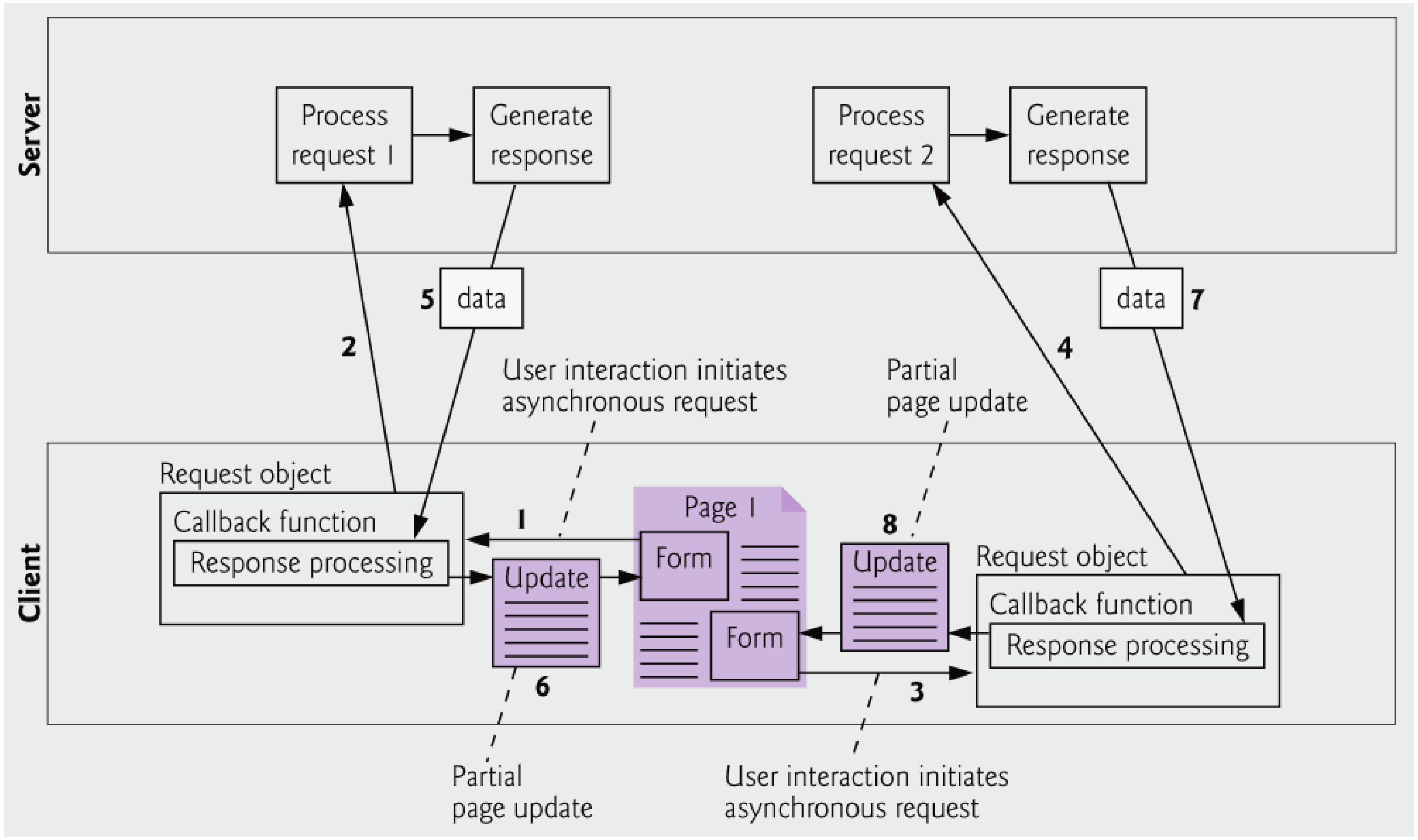
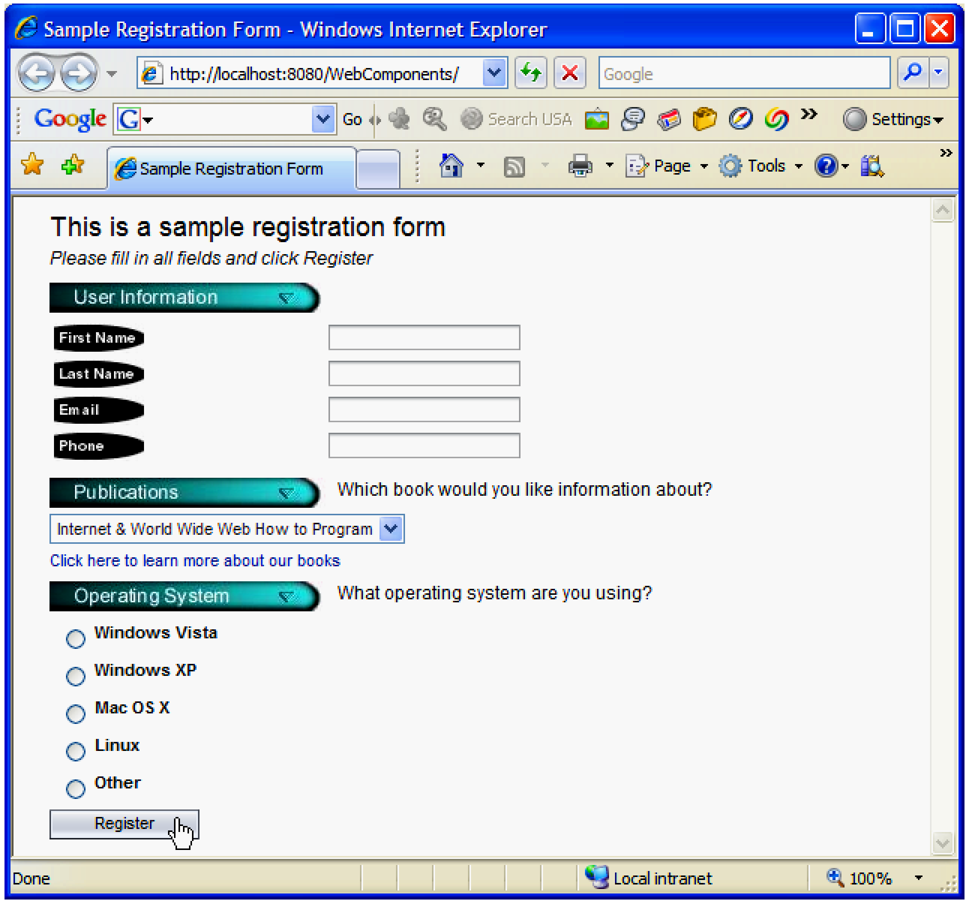


Fig. 15.2 | Ajax-enabled web application interacting with the server asynchronously.

15.3 Rich Internet Applications (RIAs) with Ajax

- **Classic XHTML registration form**
  - **Sends all of the data to be validated to the server when the user clicks the Register button**
  - **While the server is validating the data, the user cannot interact with the page**
  - **Server finds invalid data, generates a new page identifying the errors in the form and sends it back to the client—which renders the page in the browser**
  - **User fixes the errors and clicks the Register button again**
  - **Cycle repeats until no errors are found, then the data is stored on the server**
  - **Entire page reloads every time the user submits invalid data**
- **Ajax-enabled forms are more interactive**
  - **Entries are validated dynamically as the user enters data into the fields**
  - **If a problem is found, the server sends an error message that is asynchronously displayed to inform the user of the problem**
  - **Sending each entry asynchronously allows the user to address invalid entries quickly, rather than making edits and resubmitting the entire form repeatedly until all entries are valid**
  - **Asynchronous requests could also be used to fill some fields based on previous fields' values (e.g., city based on zipcode)**

a) A sample registration form in which the user has not filled in the required fields, but attempts to submit the form anyway by clicking **Register**.



Sample Registration Form - Windows Internet Explorer

http://localhost:8080/WebComponents/

This is a sample registration form

Please fill in all fields and click Register

User Information

First Name	
Last Name	
Email	
Phone	

Publications

Which book would you like information about?

Internet & World Wide Web How to Program

Click here to learn more about our books

Operating System

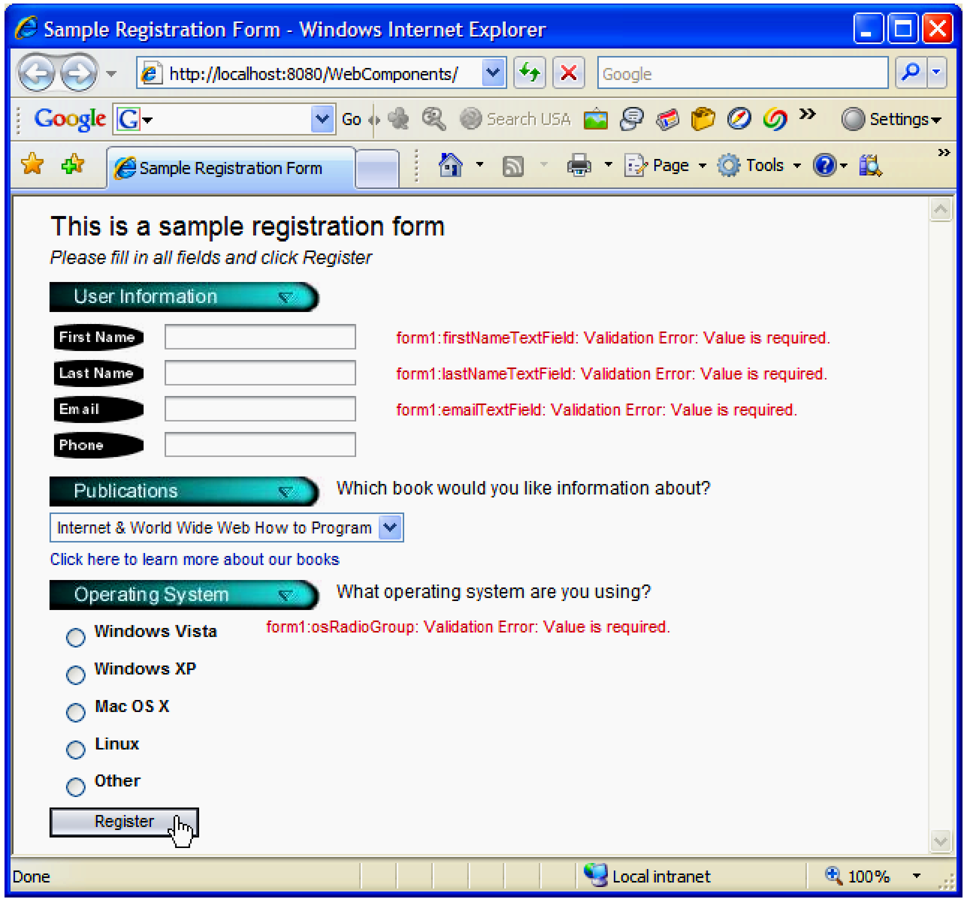
What operating system are you using?

- ○ Windows Vista
- ○ Windows XP
- ○ Mac OS X
- ○ Linux
- ○ Other

Register

Fig. 15.3 | Classic XHTML form: User submits entire form to server, which validates the data entered (if any). Server responds indicating fields with invalid or missing data. (Part 1 of 2.)

b) The server responds by indicating all the form fields with missing or invalid data. The user must correct the problems and resubmit the entire form repeatedly until all errors are corrected.


Sample Registration Form - Windows Internet Explorer
http://localhost:8080/WebComponents/
Google
Go
Search USA
Settings
Sample Registration Form
Page
Tools

This is a sample registration form

Please fill in all fields and click Register

User Information

First Name form1:firstNameTextField: Validation Error: Value is required.

Last Name form1:lastNameTextField: Validation Error: Value is required.

Email form1:emailTextField: Validation Error: Value is required.

Phone

Publications Which book would you like information about?

Internet & World Wide Web How to Program

Click here to learn more about our books

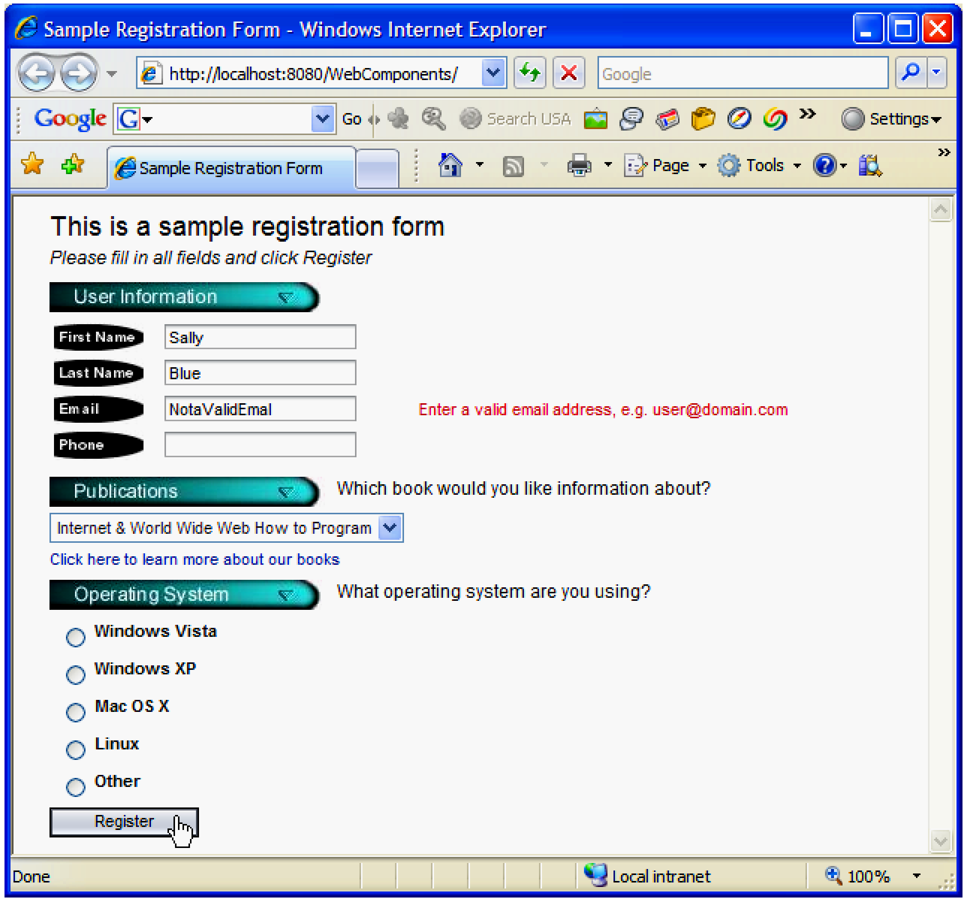
Operating System What operating system are you using?

form1:osRadioGroup: Validation Error: Value is required.

- Windows Vista
- Windows XP
- Mac OS X
- Linux
- Other

Register

Fig. 15.3 | Classic XHTML form: User submits entire form to server, which validates the data entered (if any). Server responds indicating fields with invalid or missing data. (Part 2 of 2.)


Sample Registration Form - Windows Internet Explorer

http://localhost:8080/WebComponents/

This is a sample registration form

Please fill in all fields and click Register

User Information

First Name	Sally	
Last Name	Blue	
Email	NotaValidEmal	Enter a valid email address, e.g. user@domain.com
Phone		

Publications — Which book would you like information about?

Internet & World Wide Web How to Program

Click here to learn more about our books

Operating System — What operating system are you using?

- Windows Vista
- Windows XP
- Mac OS X
- Linux
- Other

Register

Done — Local intranet — 100%

Fig. 15.4 | Ajax-enabled form shows errors asynchronously when user moves to another field.

15.4 History of Ajax

- **The term Ajax was coined by Jesse James Garrett of Adaptive Path in February 2005, when he was presenting the previously unnamed technology to a client**
- **Ajax technologies (XHTML, JavaScript, CSS, dynamic HTML, the DOM and XML) have existed for many years**
- **In 1998, Microsoft introduced the `XMLHttpRequest` object to create and manage asynchronous requests and responses**
- **Popular applications like Flickr, Google's Gmail and Google Maps use the `XMLHttpRequest` object to update pages dynamically**
- **Ajax has quickly become one of the hottest technologies in web development, as it enables webtop applications to challenge the dominance of established desktop applications**

15.5 “Raw” Ajax Example using the `XMLHttpRequest` Object

- `XMLHttpRequest` object
  - Resides on the client
  - Is the layer between the client and the server that manages asynchronous requests in Ajax applications
  - Supported on most browsers, though they may implement it differently
- To initiate an asynchronous request
  - Create an instance of the `XMLHttpRequest` object
  - Use its `open` method to set up the request, and its `send` method to initiate the request
- When an Ajax application requests a file from a server, the browser typically caches that file
  - Subsequent requests for the same file can load it from the browser’s cache
- Security
  - `XMLHttpRequest` object does not allow a web application to request resources from servers other than the one that served the web application
  - Making a request to a different server is known as cross-site scripting (also known as XSS)
  - You can implement a server-side proxy—an application on the web application’s web server—that can make requests to other servers on the web application’s behalf
- When the third argument to `XMLHttpRequest` method `open` is `true`, the request is asynchronous

Performance Tip 15.1

When an Ajax application requests a file from a server, such as an XHTML document or an image, the browser typically caches that file. Subsequent requests for the same file can load it from the browser's cache rather than making the round trip to the server again.

Software Engineering Observation 15.1

For security purposes, the `XMLHttpRequest` object doesn't allow a web application to request resources from domain names other than the one that served the application. For this reason, the web application and its resources must reside on the same web server (this could be a web server on your local computer). This is commonly known as the same origin policy (SOP). SOP aims to close a vulnerability called cross-site scripting, also known as XSS, which allows an attacker to compromise a website's security by injecting a malicious script onto the page from another domain. To learn more about XSS visit `en.wikipedia.org/wiki/XSS`. To get content from another domain securely, you can implement a server-side proxy—an application on the web application's web server—that can make requests to other servers on the web application's behalf.

15.5 "Raw" Ajax Example using the `XMLHttpRequest` Object (Cont.)

- **An exception is an indication of a problem that occurs during a program's execution**
- **Exception handling enables you to create applications that can resolve (or handle) exceptions—in some cases allowing a program to continue executing as if no problem had been encountered**
- **`try` block**
  - **Encloses code that might cause an exception and code that should not execute if an exception occurs**
  - **Consists of the keyword `try` followed by a block of code enclosed in curly braces (`{}`)**
- **When an exception occurs**
  - **`try` block terminates immediately**
  - **`catch` block catches (i.e., receives) and handles an exception**
- **`catch` block**
  - **Begins with the keyword `catch`**
  - **Followed by an exception parameter in parentheses and a block of code enclosed in curly braces**
- **Exception parameter's name**
  - **Enables the `catch` block to interact with a caught exception object, which contains `name` and `message` properties**
- **A callback function is registered as the event handler for the `XMLHttpRequest` object's `onreadystatechange` event**
  - **Whenever the request makes progress, the `XMLHttpRequest` calls the `onreadystatechange` event handler.**
  - **Progress is monitored by the `readyState` property, which has a value from 0 to 4**
  - **The value 0 indicates that the request is not initialized and the value 4 indicates that the request is complete.**

Outline

SwitchContent
ml

f 5)

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 15.5: SwitchContent.html -->
<!-- Asynchronously display content without reloading the page. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <style type="text/css">
      .box { border: 1px solid black;
             padding: 10px }
   </style>
   <title>Switch Content Asynchronously</title>
   <script type = "text/javascript" language = "JavaScript">
      <!--
      var asyncRequest; // variable to hold XMLHttpRequest objec

      // set up and send the asynchronous request
      function getContent( url )
      {
         // attempt to create the XMLHttpRequest and make the req
         try
         {
            asyncRequest = new XMLHttpRequest(); // create request object

            // register event handler
            asyncRequest.onreadystatechange = stateChange;
            asyncRequest.open( 'GET', url, true ); // prepare the request
            asyncRequest.send( null ); // send the request
         } // end try
```

The program attempts to execute the code in the `try` block. If an exception occurs, the code in the `catch` block will be executed

object and store it in `asyncRequest`

Set the event handler for the `onreadystatechange` event to the function `stateChange`

The request will be a `GET` request for the page located at `url`, and it will be asynchronous

```
            catch ( exception )
            {
               alert( 'Request failed.' );
            } // end catch
         } // end function getContent

         // displays the response data on the page
         function stateChange()
         {
            if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
            {
               document.getElementById( 'contentArea' ).innerHTML =
                  asyncRequest.responseText; // places text in contentArea
            } // end if
         } // end function stateChange

         // clear the content of the box
         function clearContent()
         {
            document.getElementById( 'contentArea' ).innerHTML = '';
         } // end function clearContent
         // -->
```

Notify the user that an error occurred

If the request has completed successfully, use the DOM to update the page with the `responseText` property of the request object

Outline

SwitchContent.html

(2 of 5)

```
   </script>
</head>
<body>
   <h1>Mouse over a book for more information.</h1>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/cpphtp6.jpg"
      onmouseover = 'getContent( "cpphtp6.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/iw3htp4.jpg"
      onmouseover = 'getContent( "iw3htp4.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/jhtp7.jpg"
      onmouseover = 'getContent( "jhtp7.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vbhtp3.jpg"
      onmouseover = 'getContent( "vbhtp3.html" )'
      onmouseout = 'clearContent()'/>
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/vcsharphtp2.jpg"
      onmouseover = 'getContent( "vcsharphtp2.html" )'
      onmouseout = 'clearContent()'/>
```

Outline

SwitchContent .html

(3 of 5)

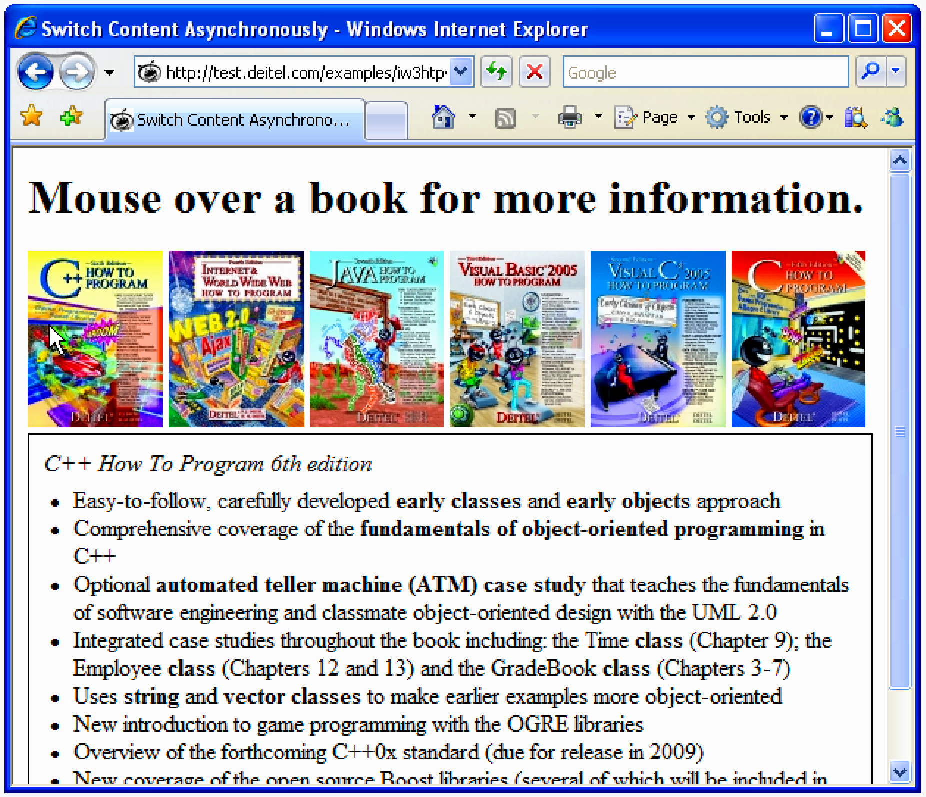
```
    <img src =
       "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/chtp5.jpg"
       onmouseover = 'getContent( "chtp5.html" )'
       onmouseout = 'clearContent()'/>
    <div class = "box" id = "contentArea">&nbsp;</div>
 </body>
 </html>
```

This `div` is updated with the description of the book that the mouse is currently hovering over

`SwitchContent.html`

(4 of 5)

a) User hovers over C++ *How to Program* book cover image, causing an asynchronous request to the server to obtain the book's description. When the response is received, the application performs a partial page update to display the description.


Switch Content Asynchronously - Windows Internet Explorer
http://test.deitel.com/examples/iw3http
Google
Switch Content Asynchrono...
Page
Tools

Mouse over a book for more information.

C++ How To Program 6th edition

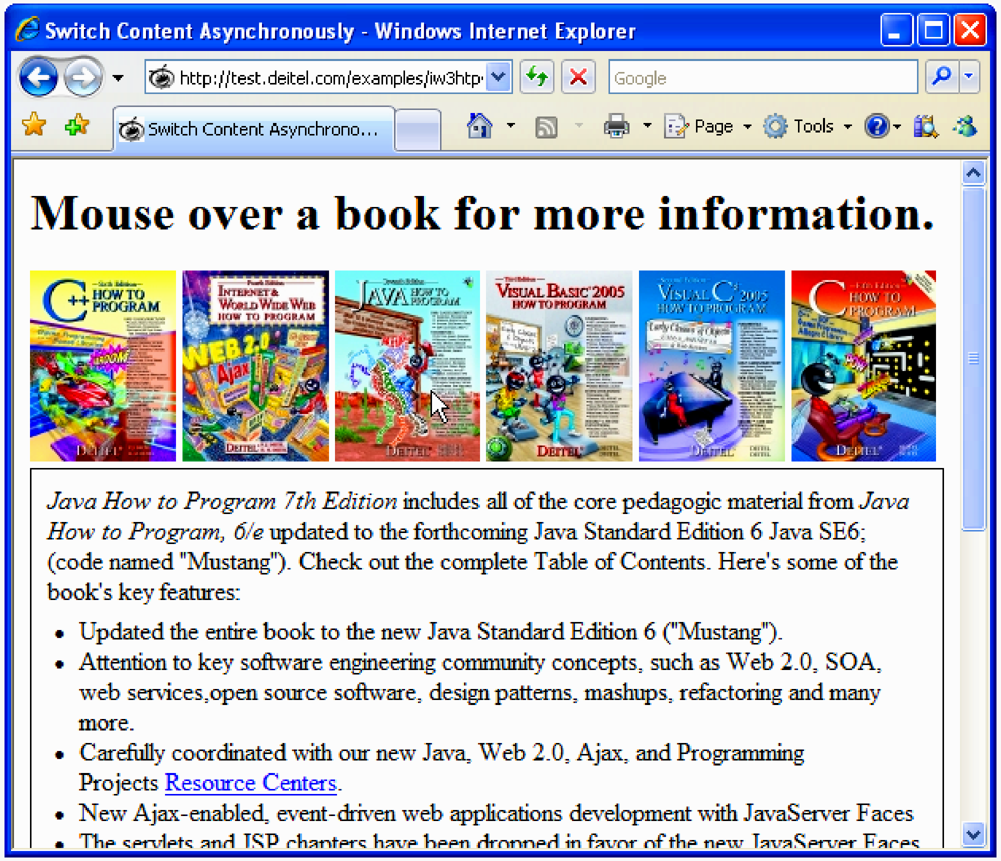
- Easy-to-follow, carefully developed **early classes** and **early objects** approach
- Comprehensive coverage of the **fundamentals of object-oriented programming** in C++
- Optional **automated teller machine (ATM) case study** that teaches the fundamentals of software engineering and classmate object-oriented design with the UML 2.0
- Integrated case studies throughout the book including: the Time **class** (Chapter 9); the Employee **class** (Chapters 12 and 13) and the GradeBook **class** (Chapters 3-7)
- Uses **string** and **vector classes** to make earlier examples more object-oriented
- New introduction to game programming with the OGRE libraries
- Overview of the forthcoming C++0x standard (due for release in 2009)
- New coverage of the open source Boost libraries (several of which will be included in

Outline

`SwitchContent .html`

(5 of 5)

b) User hovers over *Java How to Program* book cover image, causing the process to repeat.


Switch Content Asynchronously - Windows Internet Explorer
http://test.deitel.com/examples/iw3htp
Google
Switch Content Asynchrono...
Page
Tools

Mouse over a book for more information.

Java How to Program 7th Edition includes all of the core pedagogic material from *Java How to Program, 6/e* updated to the forthcoming Java Standard Edition 6 Java SE6; (code named "Mustang"). Check out the complete Table of Contents. Here's some of the book's key features:

- Updated the entire book to the new Java Standard Edition 6 ("Mustang").
- Attention to key software engineering community concepts, such as Web 2.0, SOA, web services,open source software, design patterns, mashups, refactoring and many more.
- Carefully coordinated with our new Java, Web 2.0, Ajax, and Programming Projects Resource Centers.
- New Ajax-enabled, event-driven web applications development with JavaServer Faces
- The servlets and JSP chapters have been dropped in favor of the new JavaServer Faces

Property	Description
`onreadystatechange`	Stores the callback function—the event handler that gets called when the server responds.
`readyState`	Keeps track of the request's progress. It is usually used in the callback function to determine when the code that processes the response should be launched. The `readyState` value `0` signifies that the request is uninitialized; `1` signifies that the request is loading; `2` signifies that the request has been loaded; `3` signifies that data is actively being sent from the server; and `4` signifies that the request has been completed.
`responseText`	Text that is returned to the client by the server.
`responseXML`	If the server's response is in XML format, this property contains the XML document; otherwise, it is empty. It can be used like a `document` object in JavaScript, which makes it useful for receiving complex data (e.g. populating a table).
`status`	HTTP status code of the request. A `status` of `200` means that request was successful. A `status` of `404` means that the requested resource was not found. A status of `500` denotes that there was an error while the server was proccessing the request.
`statusText`	Additional information on the request's status. It is often used to display the error to the user when the request fails.

Fig. 15.6 | **XMLHttpRequest** object properties.

Method	Description
`open`	Initializes the request and has `two` mandatory parameters—method and URL. The method parameter specifies the purpose of the request—typically `GET` if the request is to take data from the server or `POST` if the request will contain a body in addition to the headers. The URL parameter specifies the address of the file on the server that will generate the response. A third optional boolean parameter specifies whether the request is asynchronous—it's set to `true` by default.
`send`	Sends the request to the sever. It has one optional parameter, `data`, which specifies the data to be POSTed to the server—it's set to `null` by default.

Fig. 15.7 | **`XMLHttpRequest`** object methods. (Part 1 of 2.)

Method	Description
`setRequestHeader`	Alters the header of the request. The two parameters specify the header and its new value. It is often used to set the `content-type` field.
`getResponseHeader`	Returns the header data that precedes the response body. It takes one parameter, the name of the header to retrieve. This call is often used to determine the response's type, to parse the response correctly.
`getAllResponseHeaders`	Returns an array that contains all the headers that precede the response body.
`abort`	Cancels the current request.

Fig. 15.7 | `XMLHttpRequest` object methods. (Part 2 of 2.)

15.6 Using XML and the DOM

- When passing structured data between the server and the client, Ajax applications often use XML because it consumes little bandwidth and is easy to parse
- `XMLHttpRequest` object `responseXML` property
  - contains the parsed XML returned by the server
- DOM method `createElement`
  - Creates an XHTML element of the specified type
- DOM method `setAttribute`
  - Adds or changes an attribute of an XHTML element
- DOM method `appendChild`
  - Inserts one XHTML element into another
- `innerHTML` property of a DOM element
  - Can be used to obtain or change the XHTML that is displayed in a particular element

Outline

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 15.8: PullImagesOntoPage.html -->
<!-- Image catalog that uses Ajax to request XML data asynchronously. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
<title> Pulling Images onto the Page </title>
<style type = "text/css">
   td { padding: 4px }
   img { border: 1px solid black }
</style>
<script type = "text/javascript" language = "Javascript">
   var asyncRequest; // variable to hold XMLHttpRequest object

   // set up and send the asynchronous request to the XML file
   function getImages( url )
   {
      // attempt to create the XMLHttpRequest and make the request
      try
      {
         asyncRequest = new XMLHttpRequest(); // create request object

         // register event handler
         asyncRequest.onreadystatechange = processResponse;
         asyncRequest.open( 'GET', url, true ); // prepare the request
         asyncRequest.send( null ); // send the request
      } // end try
```

PullImagesOnto Page.html

(1 of 6)

Outline

```
      catch ( exception )
      {
         alert( 'Request Failed' );
      } // end catch
   } // end function getImages

   // parses the XML response; dynamically creates a table using DOM and
   // populates it with the response data; displays the table on the page
   function processResponse()
   {
      // if request completed successfully and responseXML is non-null
      if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 &&
         asyncRequest.responseXML )
      {
         clearTable(); // prepare to display a new set of images

         // get the covers from the responseXML
         var covers = asyncRequest.responseXML.getElementsByTagName(
            "cover" )

         // get base URL for the images
         var baseUrl = asyncRequest.responseXML.getElementsByTagName(
            "baseurl" ).item( 0 ).firstChild.nodeValue;

         // get the placeholder div element named covers
         var output = document.getElementById( "covers" );

         // create a table to display the images
         var imageTable = document.createElement( 'table' );

```

The `XMLHttpRequest` object's `responseXML` property contains a DOM `document` object for the loaded XML document

Get a list of the covers from the XML document

Get the base URL for the images to be displayed on the page

`PullImagesOnto Page.html`

(2 of 6)

Outline

PullImagesOntoPage.html

(3 of 6)

```
         // create the table's body
         var tableBody = document.createElement( 'tbody' );

         var rowCount = 0; // tracks number of images in current row
         var imageRow = document.createElement( "tr" ); // create row

         // place images in row
         for ( var i = 0; i < covers.length; i++ )
         {
            var cover = covers.item( i ); // get a cover from covers array

            // get the image filename
            var image = cover.getElementsByTagName( "image" ).
               item( 0 ).firstChild.nodeValue;

            // create table cell and img element to display the image
            var imageCell = document.createElement( "td" );
            var imageTag = document.createElement( "img" );

            // set img element's src attribute
            imageTag.setAttribute( "src", baseUrl + escape( image ) );
            imageCell.appendChild( imageTag ); // place img in cell
            imageRow.appendChild( imageCell ); // place cell in row
            rowCount++; // increment number of images in row

```

Insert each image based on its filename and baseurl

Outline

PullImagesOnto Page.html

(4 of 6)

```
            // if there are 6 images in the row, append the row to
            // table and start a new row
            if ( rowCount == 6 && i + 1 < covers.length )
            {
               tableBody.appendChild( imageRow );
               imageRow = document.createElement( "tr" );
               rowCount = 0;
            } // end if statement
         } // end for statement

         tableBody.appendChild( imageRow ); // append row to table body
         imageTable.appendChild( tableBody ); // append body to table
         output.appendChild( imageTable ); // append table to covers div
      } // end if
   } // end function processResponse

   // deletes the data in the table.
   function clearTable()
   {
      document.getElementById( "covers" ).innerHTML = '';
   }// end function clearTable
   </script>
</head>
<body>
   <input type = "radio" checked = "unchecked" name ="Books" value = "all"
      onclick = 'getImages( "all.xml" )'/> All Books
   <input type = "radio" checked = "unchecked"
      name = "Books" value = "simply"
      onclick = 'getImages( "simply.xml" )'/>  Simply Books
   <input type = "radio" checked = "unchecked"
      name = "Books" value = "howto"
      onclick = 'getImages( "howto.xml" )'/> How to Program Books
   <input type = "radio" checked = "unchecked"
      name = "Books" value = "dotnet"
      onclick = 'getImages( "dotnet.xml" )'/> .NET Books
   <input type = "radio" checked = "unchecked"
      name = "Books" value = "javaccpp"
      onclick = 'getImages( "javaccpp.xml" )'/> Java, C, C++ Books
   <input type = "radio" checked = "checked" name = "Books" value = "none"
      onclick = 'clearTable()'/> None
   <br/>
   <div id = "covers"></div>
</body>
</html>
```

Load the correct XML document when a radio button is clicked

Outline

PullImagesOntoPage.html

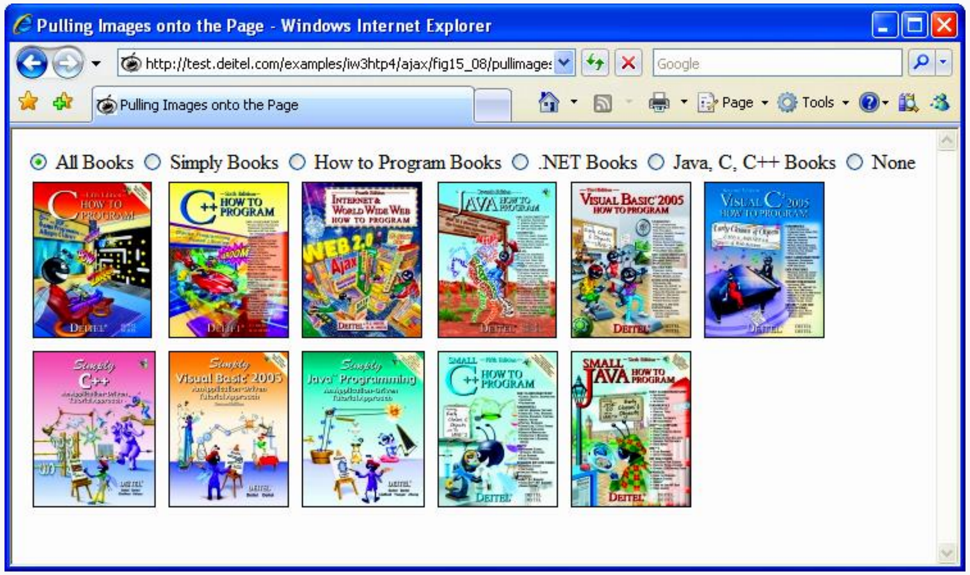
(5 of 6)

Outline

`PullImagesOnto Page.html`

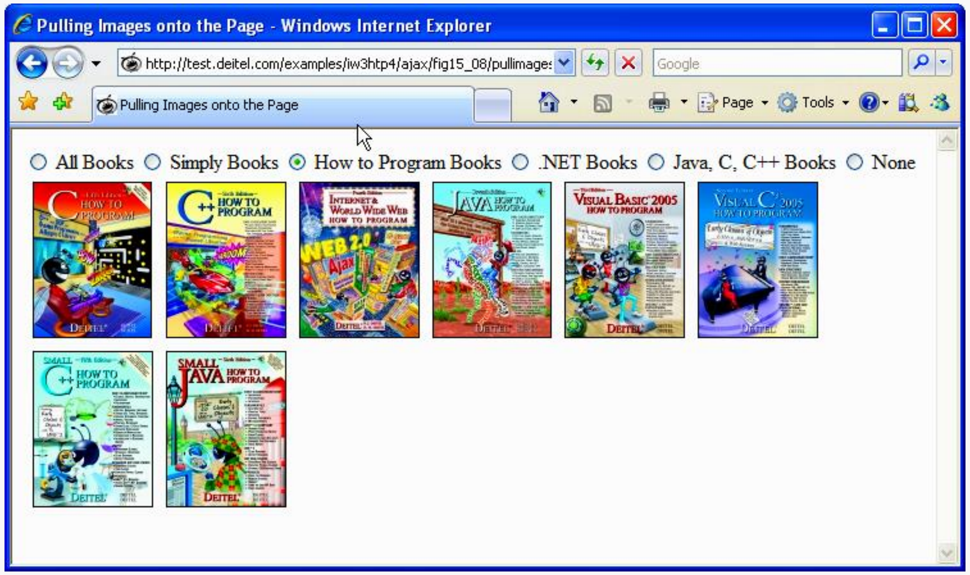
(6 of 6)

a) User clicks the **All Books** radio button to display all the book covers. The application sends an asynchronous request to the server to obtain an XML document containing the list of book-cover filenames. When the response is received, the application performs a partial page update to display the set of book covers.


Pulling Images onto the Page - Windows Internet Explorer
http://test.deitel.com/examples/iw3htp4/ajax/fig15_08/pullimage
Google
Pulling Images onto the Page
Page
Tools

[✓ All Books] Simply Books How to Program Books .NET Books Java, C, C++ Books None

b) User clicks the **How to Program Books** radio button to select a subset of book covers to display. Application sends an asynchronous request to the server to obtain an XML document containing the appropriate subset of book-cover filenames. When the response is received, the application performs a partial page update to display the subset of book covers.


Pulling Images onto the Page - Windows Internet Explorer
http://test.deitel.com/examples/iw3htp4/ajax/fig15_08/pullimage
Google
Pulling Images onto the Page
Page
Tools

All Books Simply Books [✓ How to Program Books] .NET Books Java, C, C++ Books None

15.7 Creating a Full-Scale Ajax-Enabled Application

- **JSON (JavaScript Object Notation)**
  - **Simple way to represent JavaScript objects as strings**
  - **An alternative way (to XML) to pass data between the client and the server**
- **JSON object**
  - **Represented as a list of property names and values contained in curly braces**
- **Array**
  - **Represented in JSON with square brackets containing a comma-separated list of values**
  - **Each value in a JSON array can be a string, a number, a JSON representation of an object, `true`, `false` or `null`**

15.7 Creating a Full-Scale Ajax-Enabled Application

- JavaScript `eval` function
  - Can convert JSON strings into JavaScript objects
  - To evaluate a JSON string properly, a left parenthesis should be placed at the beginning of the string and a right parenthesis at the end of the string before the string is passed to the `eval` function
  - Potential security risk—`eval` executes any embedded JavaScript code in its string argument, possibly allowing a harmful script to be injected into JSON
  - More secure way to process JSON is to use a JSON parser
- JSON strings
  - Easier to create and parse than XML
  - Require fewer bytes
  - For these reasons, JSON is commonly used to communicate in client/server interaction
- When a request is sent using the `GET` method
  - Parameters are concatenated to the URL
  - URL parameter strings start with a `?` symbol and have a list of *parameter-value* bindings, each separated by an `&`
- To implement type-ahead
  - Can use an element's `onkeyup` event handler to make asynchronous requests

Outline

addressbook.html

(1 of 18)

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 15.9 addressbook.html -->
<!-- Ajax enabled address book application. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <title>Address Book</title>
   <link rel = "stylesheet" type = "text/css" href = "address.css" />
   <script type = "text/javascript" src = "json.js"></script>
   <script type = "text/javascript">
      <!--
      // URL of the web service
      var webServiceUrl = '/AddressBookWebService/AddressService.asmx';

      var phoneValid = false; // indicates if the telephone is valid
      var zipValid = false; //indicates if the zip code is valid

      // get a list of names from the server and display them
      function showAddressBook()
      {
         // hide the "addEntry" form and show the address book
         document.getElementById( 'addEntry' ).style.display = 'none';
         document.getElementById( 'addressBook' ).style.display = 'block';

         var params = "[]"; // create an empty object
         callWebService( 'getAllNames', params, parseData );
      } // end function showAddressBook

```

Hide the form for adding an entry

Get a list of all the names from the web service, and call parseData when it's loaded

Outline

addressbook.html

(2 of 18)

```
      // send the asynchronous request to the web service
      function callWebService( method, paramString, callBack )
      {
         // build request URL string
         var requestUrl = webServiceUrl + "/" + method;
         var params = paramString.parseJSON();

         // build the parameter string to add to the url
         for ( var i = 0; i < params.length; i++ )
         {
            // checks whether it is the first parameter and builds
            // the parameter string accordingly
            if ( i == 0 )
               requestUrl = requestUrl + "?" + params[ i ].param +
                  "=" + params[ i ].value; // add first parameter to url
            else
               requestUrl = requestUrl + "&" + params[ i ].param +
                  "=" + params[ i ].value; // add other parameters to url
         } // end for

         // attempt to send the asynchronous request
         try
         {
            var asyncRequest = new XMLHttpRequest(); // create request

            // set up callback function and store it
            asyncRequest.onreadystatechange = function()
            {
               callBack( asyncRequest );
            }; // end anonymous function
```

Call a particular method by appending `method` to the base url

Parse the parameters

Build the parameter string

Set the callback function for the request to callback with the request object as a parameter

Outline

```
         // send the asynchronous request
         asyncRequest.open( 'GET', requestUrl, true );
         asyncRequest.setRequestHeader("Accept",
            "application/json; charset=utf-8" );
         asyncRequest.send(); // send request
      } // end try
      catch ( exception )
      {
         alert ( 'Request Failed' );
      } // end catch
   } // end function callWebService

   // parse JSON data and display it on the page
   function parseData( asyncRequest )
   {
      // if request has completed successfully process the response
      if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
      {
         // convert the JSON string to an Object
         var data = asyncRequest.responseText.parseJSON();
         displayNames( data ); // display data on the page
      } // end if
   } // end function parseData

```

Prepare and send the request

Parse the JSON text

`addressbook.html`

(3 of 18)

Outline

addressbook.html

(4 of 18)

```
      // use the DOM to display the retrieved address book entries
      function displayNames( data )
      {
         // get the placeholder element from the page
         var listBox = document.getElementById( 'Names' );
         listBox.innerHTML = ''; // clear the names on the page

         // iterate over retrieved entries and display them on the page
         for ( var i = 0; i < data.length; i++ )
         {
            // dynamically create a div element for each entry
            // and a fieldset element to place it in
            var entry = document.createElement( 'div' );
            var field = document.createElement( 'fieldset' );
            entry.onclick = handleOnClick; // set onclick event handler
            entry.id = i; // set the id
            entry.innerHTML = data[ i ].First + ' ' + data[ i ].Last;
            field.appendChild( entry ); // insert entry into the field
            listBox.appendChild( field ); // display the field
          } // end for
      } // end function displayAll

      // event handler for entry's onclick event
      function handleOnClick()
      {
         // call getAddress with the element's content as a parameter
         getAddress( eval( 'this' ), eval( 'this.innerHTML' ) );
      } // end function handleOnClick

```

Create an XHTML fieldset element for the entry

Insert the name into the entry

Use `this` to give the clicked element the correct parameters

Outline

```
      // search the address book for input
      // and display the results on the page
      function search( input )
      {
         // get the placeholder element and delete its content
         var listBox = document.getElementById( 'Names' );
         listBox.innerHTML = ''; // clear the display box

         // if no search string is specified all the names are displayed
         if ( input == "" ) // if no search value specified
         {
            showAddressBook(); // Load the entire address book
         } // end if
         else
         {
            var params = '[{"param": "input", "value": "' + input + '"}]';
            callWebService( "search",  params , parseData );
         } // end else
      } // end function search

      // Get address data for a specific entry
      function getAddress( entry, name )
      {
         // find the address in the JSON data using the element's id
         // and display it on the page
         var firstLast = name.split(" "); // convert string to array
         var requestUrl = webServiceUrl + "/getAddress?first="
            + firstLast[ 0 ] + "&last=" + firstLast[ 1 ];

```

addressbook.html

(5 of 18)

Create a JSON parameter object to search for `input` in the list of names

Assemble the web service call

```
         // attempt to send an asynchronous request
         try
         {
            // create request object
            var asyncRequest = new XMLHttpRequest();

            // create a callback function with 2 parameters
            asyncRequest.onreadystatechange = function()
            {
               displayAddress( entry, asyncRequest );
            }; // end anonymous function

            asyncRequest.open( 'GET', requestUrl, true );
            asyncRequest.setRequestHeader("Accept",
            "application/json; charset=utf-8"); // set response datatype
            asyncRequest.send(); // send request
         } // end try
         catch ( exception )
         {
            alert ( 'Request Failed.' );
         } // end catch
      } // end function getAddress

```

Outline

addressbook.html

(6 of 18)

Outline

addressbook.html

(7 of 18)

```
      // clear the entry's data.
      function displayAddress( entry, asyncRequest )
      {
         // if request has completed successfully, process the response
         if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
         {
            // convert the JSON string to an object
            var data = asyncRequest.responseText.parseJSON();
            var name = entry.innerHTML // save the name string
            entry.innerHTML = name + '<br/>' + data.Street +
               '<br/>' + data.City + ', ' + data.State
               + ', ' + data.Zip + '<br/>' + data.Telephone;

            // clicking on the entry removes the address
            entry.onclick = function()
            {
               clearField( entry, name );
            }; // end anonymous function

          } // end if
      } // end function displayAddress

```

Parse and display the address details for an entry

Outline

```
    // clear the entry's data
    function clearField( entry, name )
    {
       entry.innerHTML = name; // set the entry to display only the name
       entry.onclick = function() // set onclick event
       {
          getAddress( entry, name ); // retrieve address and display it
       }; // end function
    } // end function clearField

    // display the form that allows the user to enter more data
    function addEntry()
    {
       document.getElementById( 'addressBook' ).style.display = 'none';
       document.getElementById( 'addEntry' ).style.display = 'block';
    } // end function addEntry

    // send the zip code to be validated and to generate city and state
    function validateZip( zip )
    {
       // build parameter array
       var params = '[{"param": "zip", "value": "' + zip + '"}]';
       callWebService ( "validateZip", params, showCityState );
    } // end function validateZip

```

addressbook.html

(8 of 18)

Clear the address from an entry and reset the event handler to display the address again when clicked

Make the zip-code validation web service call

Outline

Notify the user that the zip code is being checked

addressbook.html

(9 of 18)

```
      // get city and state that were generated using the zip code
      // and display them on the page
      function showCityState( asyncRequest )
      {
         // display message while request is being processed
         document.getElementById( 'validateZip' ).
            innerHTML = "Checking zip...";

         // if request has completed successfully, process the response
         if ( asyncRequest.readyState == 4 )
         {
            if ( asyncRequest.status == 200 )
            {
               // convert the JSON string to an object
               var data = asyncRequest.responseText.parseJSON();

               // update zip code validity tracker and show city and state
               if ( data.Validity == 'Valid' )
               {
                  zipValid = true; // update validity tracker

                  // display city and state
                  document.getElementById( 'validateZip' ).innerHTML = '';
                  document.getElementById( 'city' ).innerHTML = data.City;
                  document.getElementById( 'state' ).
                     innerHTML = data.State;
               } // end if
```

```
                else
                {
                   zipValid = false; // update validity tracker
                   document.getElementById( 'validateZip' ).
                      innerHTML = data.ErrorText; // display the error

                   // clear city and state values if they exist
                   document.getElementById( 'city' ).innerHTML = '';
                   document.getElementById( 'state' ).innerHTML = '';
                } // end else
             } // end if
             else if ( asyncRequest.status == 500 )
             {
                document.getElementById( 'validateZip' ).
                   innerHTML = 'Zip validation service not avaliable';
             } // end else if
          } // end if
       } // end function showCityState

       // send the telephone number to the server to validate format
       function validatePhone( phone )
       {
          var params = '[{ "param": "tel", "value": "' + phone + '"}]';
          callWebService( "validateTel", params, showPhoneError );
       } // end function validatePhone

```

addressbook.html

(10 of 18)

```
      // show whether the telephone number has correct format
      function showPhoneError( asyncRequest )
      {
         // if request has completed successfully, process the response
         if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
         {
            // convert the JSON string to an object
            var data = asyncRequest.responseText.parseJSON();

            if ( data.ErrorText != "Valid Telephone Format" )
            {
               phoneValid = false; // update validity tracker
            } // end if
            else
            {
               phoneValid = true; // update validity tracker
            } // end else

            document.getElementById( 'validatePhone' ).
               innerHTML = data.ErrorText; // display the error
          } // end if
      } // end function showPhoneError

      // enter the user's data into the database
```

Outline

addressbook.html

(11 of 18)

Get the response from the request object so we can determine if the phone number is valid

```
      function saveForm()
      {
         // retrieve the data from the form
         var first = document.getElementById( 'first' ).value;
         var last = document.getElementById( 'last' ).value;
         var street = document.getElementById( 'street' ).value;
         var city = document.getElementById( 'city' ).innerHTML;
         var state = document.getElementById( 'state' ).innerHTML;
         var zip = document.getElementById( 'zip' ).value;
         var phone = document.getElementById( 'phone' ).value;

         // check if data is valid
         if ( !zipValid || !phoneValid  )
         {
            // display error message
            document.getElementById( 'success' ).innerHTML =
               'Invalid data entered. Check form for more information';
         } // end if
         else if ( ( first == "" ) || ( last == "" ) )
         {
            // display error message
            document.getElementById( 'success').innerHTML =
               'First Name and Last Name must have a value.';
         } // end if
```

Outline

addressbook.html

(12 of 18)

```
         else
         {
            // hide the form and show the addressbook
            document.getElementById( 'addEntry' )
               .style.display = 'none';
            document.getElementById( 'addressBook' ).
               style.display = 'block';

            // build the parameter to include in the web service URL
            params = '[{"param": "first", "value": "' + first +
               '"}, { "param": "last", "value": "' + last +
               '"}, { "param": "street", "value": "'+ street +
               '"}, { "param": "city", "value": "' +  city +
               '"}, { "param":  "state", "value:": "' + state +
               '"}, { "param": "zip", "value": "' + zip +
               '"}, { "param": "tel", "value": "' + phone + '"}]';

            // call the web service to insert data into the database
            callWebService( "addEntry", params, parseData );
         } // end else
      } // end function saveForm
      //-->
```

Outline

addressbook.html

(13 of 18)

Create a JSON object with all the data as paramters for the web service call to save the data

```
   </script>
</head>
<body onload = "showAddressBook()">
   <div>
      <input type = "button" value = "Address Book"
         onclick = "showAddressBook()"/>
      <input type = "button" value = "Add an Entry"
         onclick = "addEntry()"/>
   </div>
   <div id = "addressBook" style = "display : block;">
       Search By Last Name:
      <input onkeyup = "search( this.value )"/>
      <br/>
      <div id = "Names">
      </div>
   </div>
   <div id = "addEntry" style = "display : none">
      First Name: <input id = 'first'/>
      <br/>
      Last Name: <input id = 'last'/>
      <br/>
      <strong> Address: </strong>
      <br/>
      Street: <input id = 'street'/>
      <br/>
      City: <span id = "city" class = "validator"></span>
      <br/>
      State: <span id = "state" class = "validator"></span>
      <br/>
      Zip: <input id = 'zip' onblur = 'validateZip( this.value )'/>
```

The search is done for every onkeyup event, giving the form live search functionality

Outline

addressbook.html

(14 of 18)

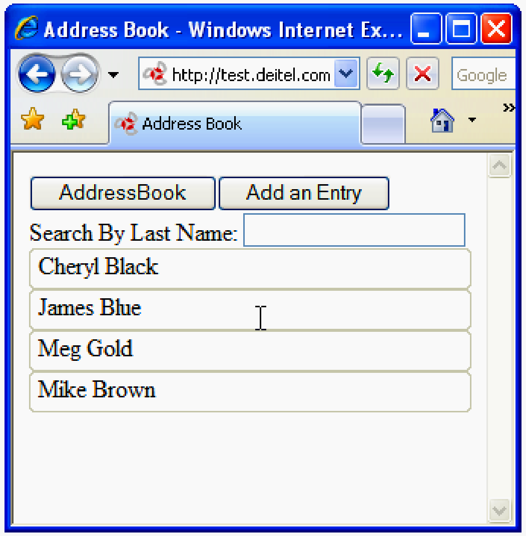
Outline

addressbook.html

(15 of 18)

```
      <span id = "validateZip" class = "validator">
      </span>
      <br/>
      Telephone:<input id = 'phone'
         onblur = 'validatePhone( this.value )'/>
      <span id = "validatePhone" class = "validator">
      </span>
      <br/>
      <input type = "button" value = "Submit"
         onclick = "saveForm()" />
      <br/>
      <div id = "success" class = "validator">
      </div>
   </div>
</body>
</html>
```

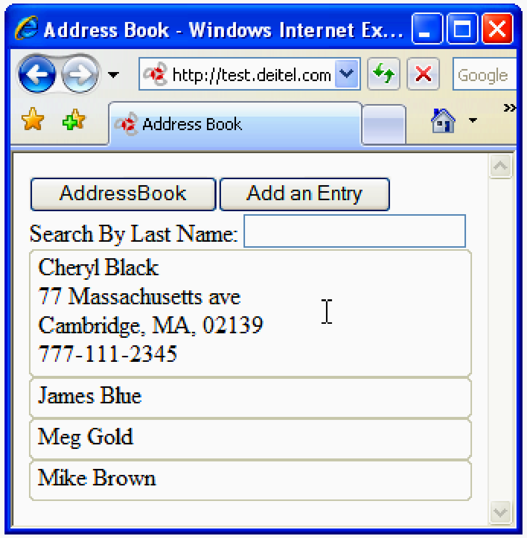
a) Page is loaded. All the entries are displayed.



Search By Last Name:

Cheryl Black
James Blue
Meg Gold
Mike Brown

b) User clicks on an entry. The entry expands, showing the address and the telephone.



Search By Last Name:

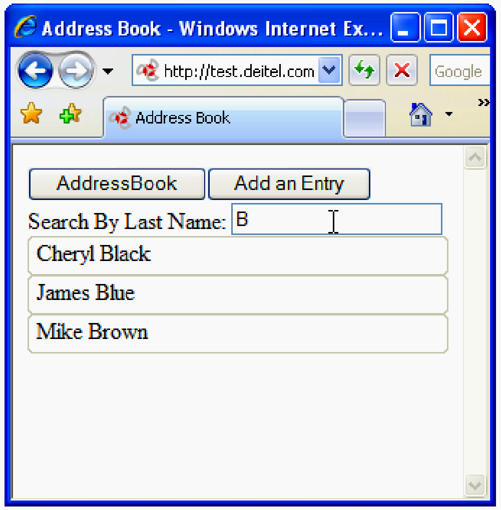
Cheryl Black
77 Massachusetts ave
Cambridge, MA, 02139
777-111-2345
James Blue
Meg Gold
Mike Brown

Outline

addressbook.html

(16 of 18)

c) User types "B" in the search field. Application loads the entries whose last names start with "B".



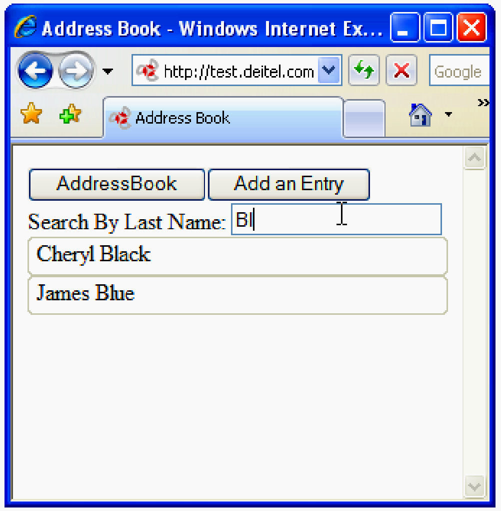
Search By Last Name: B

Cheryl Black

James Blue

Mike Brown

d) User types "Bl" in the search field. Application loads the entries whose last names start with "Bl".

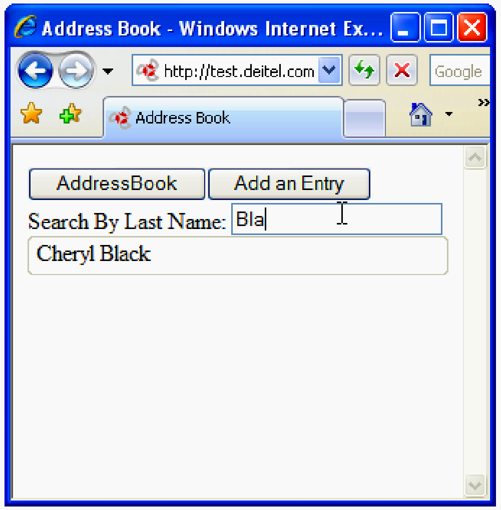


Search By Last Name: Bl

Cheryl Black

James Blue

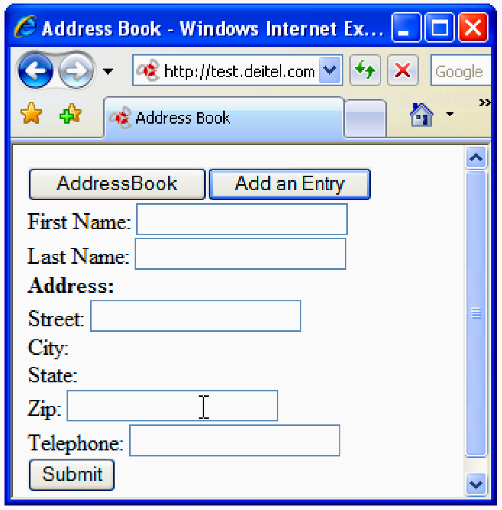
e) User types "Bla" in the search field. Application loads the entries whose last names start with "Bla".



Search By Last Name: Bla

Cheryl Black

f) User clicks **Add an Entry** button. The form allowing user to add an entry is displayed.



First Name:

Last Name:

Address:

Street:

City:

State:

Zip:

Telephone:

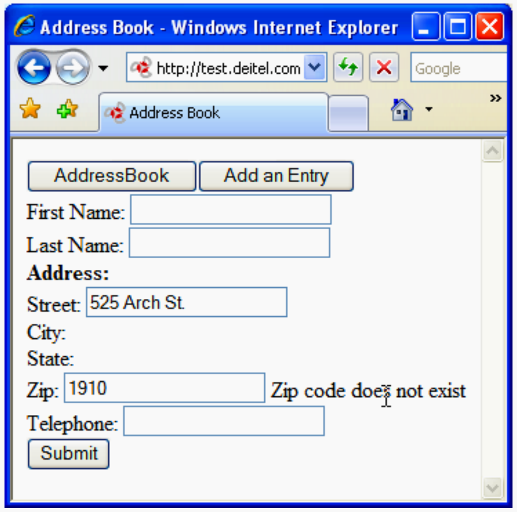
Submit

Outline

addressbook.html

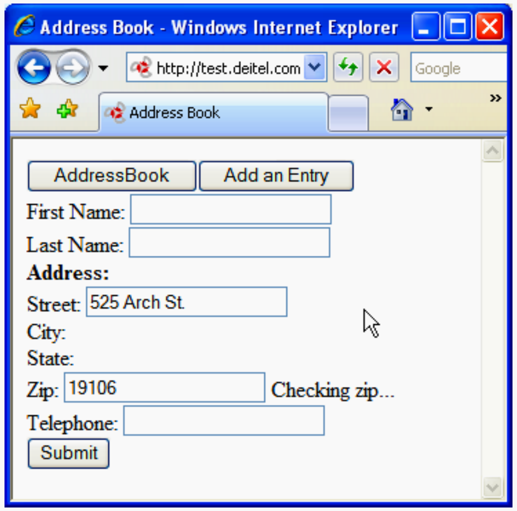
(17 of 18)

g) User types in a nonexistent zip code. An error is displayed.



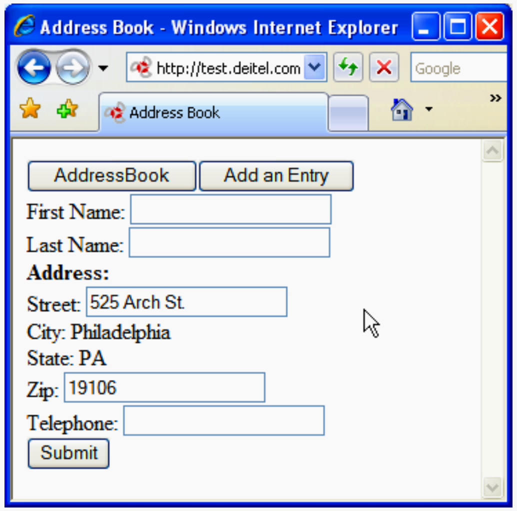
First Name:
Last Name:
Address:
Street: 525 Arch St.
City:
State:
Zip: 1910 Zip code does not exist
Telephone:

h) User enters a valid zip code. While the server processes it, **Checking Zip...** is displayed on the page.



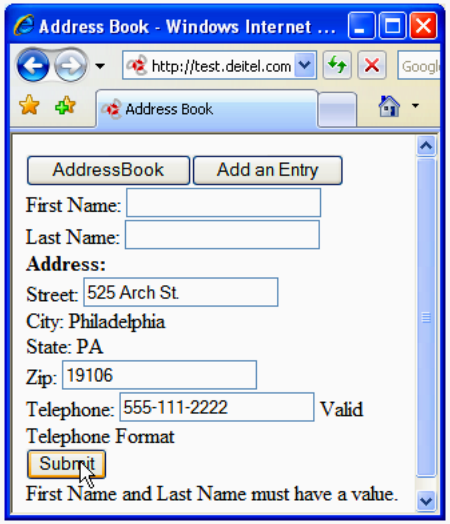
First Name:
Last Name:
Address:
Street: 525 Arch St.
City:
State:
Zip: 19106 Checking zip...
Telephone:

i) The server finds the city and state associated with the zip code entered and displays them on the page.



First Name:
Last Name:
Address:
Street: 525 Arch St.
City: Philadelphia
State: PA
Zip: 19106
Telephone:

j) The user enters a telephone number and tries to submit the data. The application does not allow this, because the First Name and Last Name are empty.



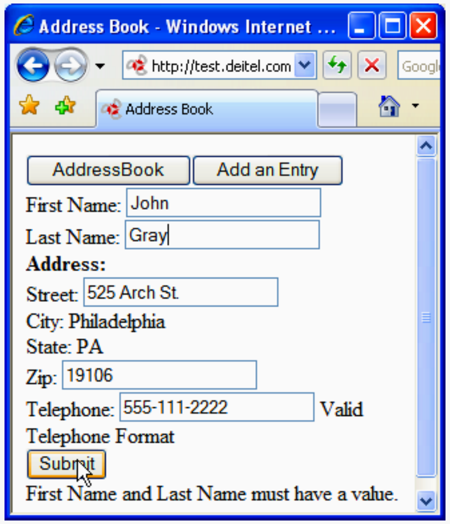
First Name:
Last Name:
Address:
Street: 525 Arch St.
City: Philadelphia
State: PA
Zip: 19106
Telephone: 555-111-2222 Valid Telephone Format
First Name and Last Name must have a value.

Outline

addressbook.html

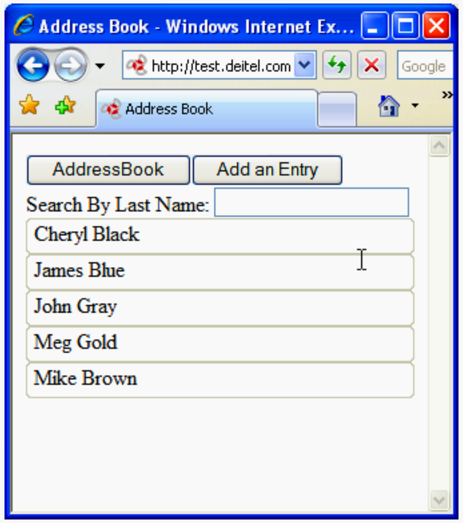
(18 of 18)

k) The user enters the last name and the first name and clicks the Submit button.



First Name: John
Last Name: Gray
Address:
Street: 525 Arch St
City: Philadelphia
State: PA
Zip: 19106
Telephone: 555-111-2222 Valid Telephone Format
First Name and Last Name must have a value.

l) The address book is redisplayed with the new name added in.



Search By Last Name:
Cheryl Black
James Blue
John Gray
Meg Gold
Mike Brown

Outline

```
[ { "first": "Cheryl", "last": "Black" },
  { "first": "James", "last": "Blue" },
  { "first": "Mike", "last": "Brown" },
  { "first": "Meg", "last": "Gold" } ]
```

15.8 Dojo Toolkit

- **Cross-browser compatibility, DOM manipulation and event handling can be cumbersome, particularly as an application's size increases**
- **Dojo**
  - **Free, open source JavaScript library that simplifies Ajax development**
  - **Reduces asynchronous request handling to a single function call**
  - **Provides cross-browser DOM functions that simplify partial page updates**
  - **Provides event handling and rich GUI controls**
- **To install Dojo**
  - **Download the latest release from `www.Dojotoolkit.org/downloads` to your hard drive**
  - **Extract the files from the archive file to your web development directory or web server**
  - **To include the `Dojo.js` script file in your web application, place the following script in the head element of your XHTML document:**

    ```
    <script type = "text/javascript" src = "path/Dojo.js">
    ```

    where *path* is the relative or complete path to the Dojo toolkit's files
- **Edit-in-place**
  - **Enables a user to modify data directly in the web page**
  - **Common feature in Ajax applications**

15.8 Dojo Toolkit (Cont.)

- **Dojo is organized in packages of related functionality**
- **`dojo.require` method**
  - **Used to include specific Dojo packages**
- **`dojo.io` package functions communicate with the server**
- **`dojo.event` package simplifies event handling**
- **`dojo.widget` package provides rich GUI controls**
- **`dojo.dom` package contains DOM functions that are portable across many different browsers**
- **Dojo widget**
  - **Any predefined user interface element that is part of the Dojo toolkit**
  - **To incorporate an existing widget onto a page, set the `dojoType` attribute of any HTML element to the type of widget that you want it to be**
- **`dojo.widget.byId` method can be used to obtain a Dojo widget**
- **`dojo.events.connect` method links functions together**
- **`dojo.date.toRfc3339` method converts a date to *yyyy-mm-dd* format**

15.8 Dojo Toolkit (Cont.)

- `dojo.io.bind` method
  - Configures and sends asynchronous requests
  - takes an array of parameters, formatted as a JavaScript object
  - `url` parameter specifies the destination of the request
  - `handler` parameter specifies the callback function (which must have parameters parameters—`type`, `data` and `event`)
  - `mimetype` parameter specifies the format of the response
  - `handler` parameter can be replaced by the `load` and `error` parameters
  - The function passed as the `load` handler processes successful requests
  - The function passed as the `error` handler processes unsuccessful requests
- Dojo calls the function passed as the `handler` parameter only when the request completes
- In Dojo, the script does not have access to the request object. All the response data is sent directly to the callback function.
- `dojo.dom.insertAtIndex` method
  - Inserts an element at the specified index in the DOM
- `removeNode` function removes an element from the DOM
- Dojo Button widgets use their own `buttonClick` event instead of the DOM `onclick` event to store the event handler
- `Event` object's `currentTarget` property contains the element that initiated the event

Outline

Calendar.html

(1 of 11)

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 15.11 Calendar.html -->
<!-- Calendar application built with dojo. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
<head>
   <script type = "text/javascript" src = "/dojo043/dojo.js"></script>
   <script type = "text/javascript" src = "json.js"></script>
   <script type = "text/javascript">
      <!--
      // specify all the required dojo scripts
      dojo.require( "dojo.event.*" ); // use scripts from event package
      dojo.require( "dojo.widget.*" ); // use scripts from widget package
      dojo.require( "dojo.dom.*" ); // use scripts from dom package
      dojo.require( "dojo.io.*" ); // use scripts from the io package

      // configure calendar event handler
      function connectEventHandler()
      {
         var calendar = dojo.widget.byId( "calendar" ); // get calendar
         calendar.setDate( "2007-07-04" );
         dojo.event.connect(
            calendar, "onValueChanged", "retrieveItems" );
      } // end function connectEventHandler

```

Import dojo packages

Get the calendar, set the date, and connect an event handler so that `retrieveItems` is called when the calendar is changed

Outline

`Calendar.html`

(2 of 11)

```
   // location of CalendarService web service
   var webServiceUrl = "/CalendarService/CalendarService.asmx";

   // obtain scheduled events for the specified date
   function retrieveItems( eventDate )
   {
      // convert date object to string in yyyy-mm-dd format
      var date = dojo.date.toRfc3339( eventDate ).substring( 0, 10 );

       // build parameters and call web service
      var params = '[{ "param":"eventDate", "value":"' +
         date +  "'}]";
      callWebService( 'getItemsByDate', params, displayItems );
   } // end function retrieveItems

    // call a specific web service asynchronously to get server data
   function callWebService( method, params, callback )
   {
      // url for the asynchronous request
      var requestUrl = webServiceUrl + "/" + method;
      var params = paramString.parseJSON();

      // build the parameter string to append to the url
      for ( var i = 0; i < params.length; i++ )
      {
         // check if it is the first parameter and build
         // the parameter string accordingly
         if ( i == 0 )
            requestUrl = requestUrl + "?" + params[ i ].param +
               "=" + params[ i ].value; // add first parameter to url
```

Build a JSON object to hold the date parameter and call the web service

Import dojo packages

```
            else
               requestUrl = requestUrl + "&" + params[ i ].param +
                  "=" + params[ i ].value; // add other parameters to url
         } // end for

         // call asynchronous request using dojo.io.bind
         dojo.io.bind( { url: requestUrl, handler: callback,
            accept: "application/json; charset=utf-8" } );
      } // end function callWebService

      // display the list of scheduled events on the page
      function displayItems( type, data, event )
      {
         if ( type == 'error' ) // if the request has failed
         {
            alert( 'Could not retrieve the event' ); // display error
         } // end if
         else
         {
            var placeholder = dojo.byId( "itemList" ); // get placeholder
            placeholder.innerHTML = ''; // clear placeholder
            var items = data.parseJSON(); // parse server data

            // check whether there are events;
            // if none then display message
            if ( items == "" )
            {
               placeholder.innerHTML = 'No events for this date.';
            }

```

Outline

`Calendar.html`

(3 of 11)

`dojo.io.bind` makes a call, sets a callback handler, and specifies what type of data to accept

Get the placeholder by its `id`

Outline

`Calendar.html`

(4 of 11)

```
            for ( var  i = 0; i < items.length; i++ )
            {
               // initialize item's container
               var item = document.createElement( "div" );
               item.id = items[ i ].id; // set DOM id to database id

               // obtain and paste the item's description
               var text = document.createElement( "div" );
               text.innerHTML =  items[i].description;
               text.id = 'description' + item.id;
               dojo.dom.insertAtIndex( text, item, 0 );

               // create and insert the placeholder for the edit button
               var buttonPlaceHolder = document.createElement( "div" );
               dojo.dom.insertAtIndex( buttonPlaceHolder, item, 1 );

               // create the edit button and paste it into the container
               var editButton = dojo.widget.
                  createWidget( "Button", {}, buttonPlaceHolder );
               editButton.setCaption( "Edit" );
               dojo.event.connect(
                  editButton, 'buttonClick', handleEdit );

               // insert item container in the list of items container
               dojo.dom.insertAtIndex( item, placeholder, i );
            } // end for
         } // end else
      } // end function displayItems

```

Use dojo's cross-browser DOM features to update the page

Create an edit button for the event

Outline

`Calendar.html`

(5 of 11)

```
      // send the asynchronous request to get content for editing and
      // run the edit-in-place UI
      function handleEdit( event )
      {
         var id = event.currentTarget.parentNode.id; // retrieve id
         var params = '[{ "param":"id", "value":"' + id +  '"}]';
         callWebService( 'getItemById', params, displayForEdit );
      } // end function handleEdit

      // set up the interface for editing an item
      function displayForEdit(type, data, event)
      {
         if ( type == 'error' ) // if the request has failed
         {
            alert( 'Could not retrieve the event' ); // display error
         }
         else
         {
            var item = data.parseJSON(); // parse the item
            var id = item.id; // set the id

            // create div elements to insert content
            var editElement = document.createElement( 'div' );
            var buttonElement = document.createElement( 'div' );

            // hide the unedited content
            var oldItem = dojo.byId( id ); // get the original element
            oldItem.id = 'old' + oldItem.id; // change element's id
            oldItem.style.display = 'none'; // hide old element
            editElement.id = id; // change the "edit" container's id
```

Call the web service to get the item to be edited

Outline

```

      // create a textbox and insert it on the page
      var editArea = document.createElement( 'textarea' );
      editArea.id = 'edit' + id; // set textbox id
      editArea.innerHTML = item.description; // insert description
      dojo.dom.insertAtIndex( editArea, editElement, 0 );

      // create button placeholders and insert on the page
      // these will be transformed into dojo widgets
      var saveElement = document.createElement( 'div' );
      var cancelElement = document.createElement( 'div' );
      dojo.dom.insertAtIndex( saveElement, buttonElement, 0 );
      dojo.dom.insertAtIndex( cancelElement, buttonElement, 1 );
      dojo.dom.insertAtIndex( buttonElement, editElement, 1 );

      // create "save" and "cancel" buttons
      var saveButton =
         dojo.widget.createWidget( "Button", {}, saveElement );
      var cancelButton =
         dojo.widget.createWidget( "Button", {}, cancelElement );
      saveButton.setCaption( "Save" ); // set saveButton label
      cancelButton.setCaption( "Cancel" ); // set cancelButton text

      // set up the event handlers for cancel and save buttons
      dojo.event.connect( saveButton, 'buttonClick', handleSave );
      dojo.event.connect(
         cancelButton, 'buttonClick', handleCancel );
```

Calendar.html

(6 of 11)

Outline

Calendar.html

(7 of 11)

```
            // paste the edit UI on the page
            dojo.dom.insertAfter( editElement, oldItem );
         } // end else
      } // end function displayForEdit

      // sends the changed content to the server to be saved
      function handleSave( event )
      {
         // grab user entered data
         var id = event.currentTarget.parentNode.parentNode.id;
         var descr = dojo.byId( 'edit' + id ).value;

         // build parameter string and call the web service
         var params = '[{ "param":"id", "value":"' + id +
            '"}, {"param": "descr", "value":"' + descr + '"}]';
         callWebService( 'Save', params, displayEdited );
      } // end function handleSave

      // restores the original content of the item
      function handleCancel( event )
      {
         var voidEdit = event.currentTarget.parentNode.parentNode;
         var id = voidEdit.id; // retrieve the id of the item
         dojo.dom.removeNode( voidEdit, true ); // remove the edit UI
         var old = dojo.byId( 'old' + id ); // retrieve pre-edit version
         old.style.display = 'block'; // show pre-edit version
         old.id = id; // reset the id
      } // end function handleCancel

```

Cancel the edit by removing the form and showing the old item

Outline

Calendar.html

(8 of 11)

```
   // displays the updated event information after an edit is saved
   function displayEdited( type, data, event )
   {
      if ( type == 'error' )
      {
         alert( 'Could not retrieve the event' );
      }
      else
      {
         editedItem = data.parseJSON(); // obtain updated description
         var id = editedItem.id; // obtain the id
         var editElement = dojo.byId( id ); // get the edit UI
         dojo.dom.removeNode( editElement, true ); // delete edit UI
         var old = dojo.byId( 'old' + id ); // get item container

         // get pre-edit element and update its description
         var oldText = dojo.byId( 'description' + id );
         oldText.innerHTML = editedItem.description;

         old.id = id; // reset id
         old.style.display = 'block'; // show the updated item
      } // end else
   } // end function displayEdited

   // when the page is loaded, set up the calendar event handler
   dojo.addOnLoad( connectEventHandler );
   // -->
```

Set `connectEventHandler` to be called `onload`

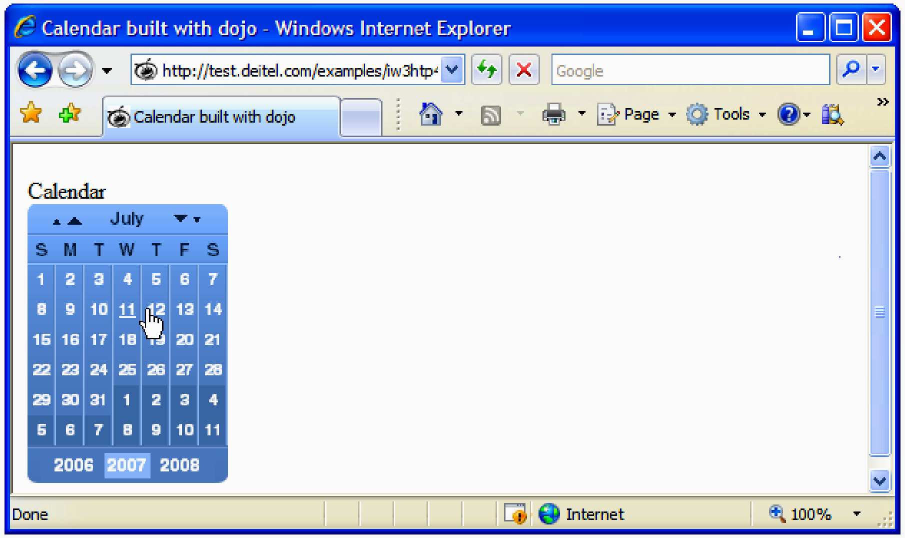
Outline

Calendar.html

(9 of 11)

```
   </script>
   <title> Calendar built with dojo </title>
</head>
<body>
   Calendar
   <div dojoType = "datePicker" style = "float: left"
      widgetID = "calendar"></div>
   <div id = "itemList" style = "float: left"></div>
</body>
</html>
```

a) DatePicker Dojo widget after the web page loads.

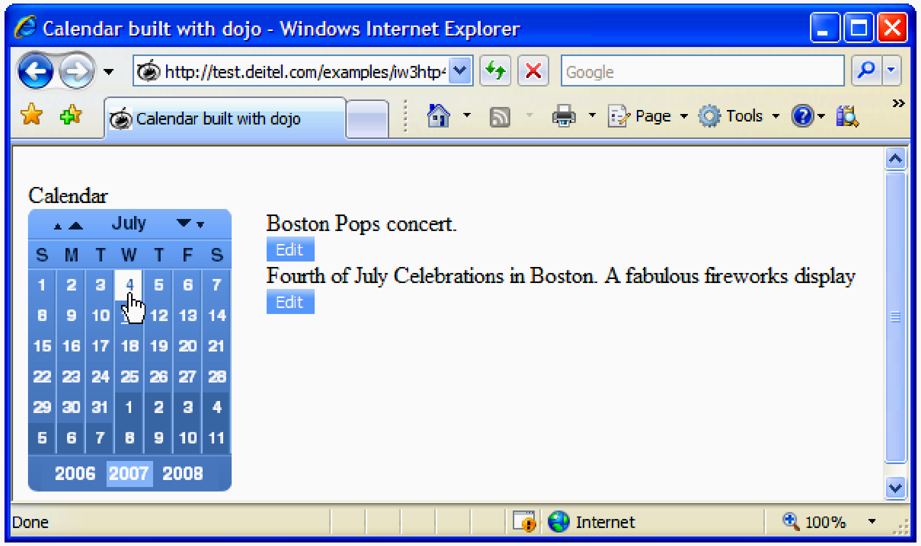


Outline

Calendar.html

(10 of 11)

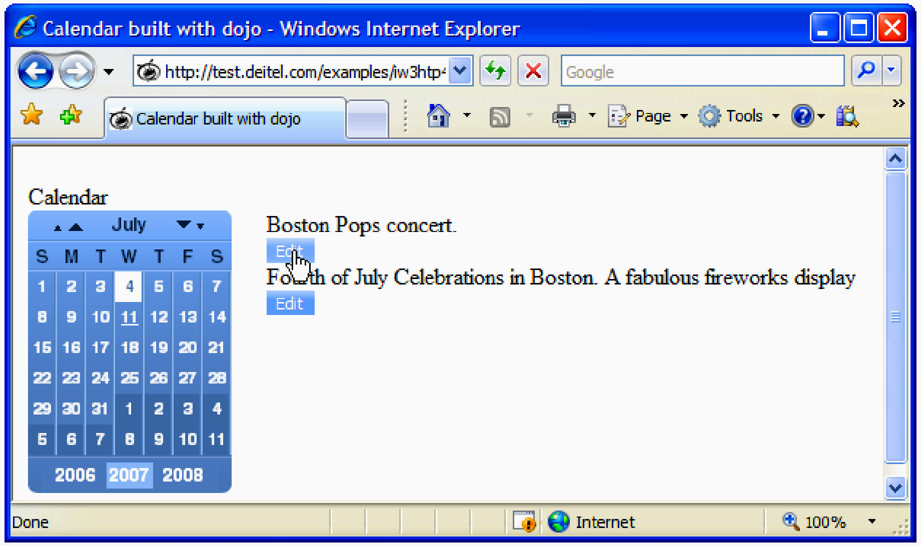
b) User selects a date and the application asynchronously requests a list of events for that date and displays the results with a partial page update.


Calendar built with dojo - Windows Internet Explorer
http://test.deitel.com/examples/iw3htp4
Calendar built with dojo
Calendar
July
2006 2007 2008
Edit
Done
Internet
100%

Boston Pops concert.

Fourth of July Celebrations in Boston. A fabulous fireworks display

c) User clicks the **Edit** button to modify an event's description.


Calendar built with dojo - Windows Internet Explorer
http://test.deitel.com/examples/iw3htp4
Calendar built with dojo
Calendar
July
2006 2007 2008
Edit
Done
Internet
100%

Boston Pops concert.

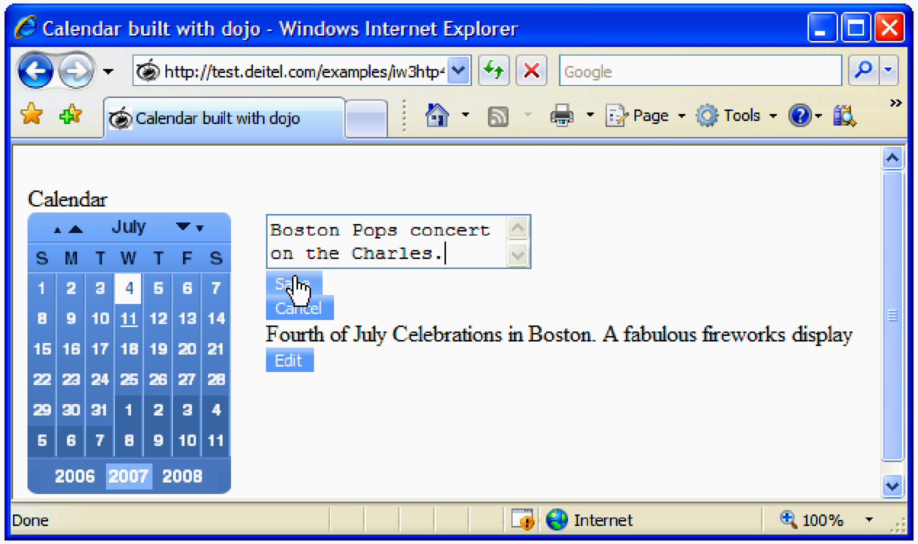
Fourth of July Celebrations in Boston. A fabulous fireworks display

Outline

Calendar.html

(11 of 11)

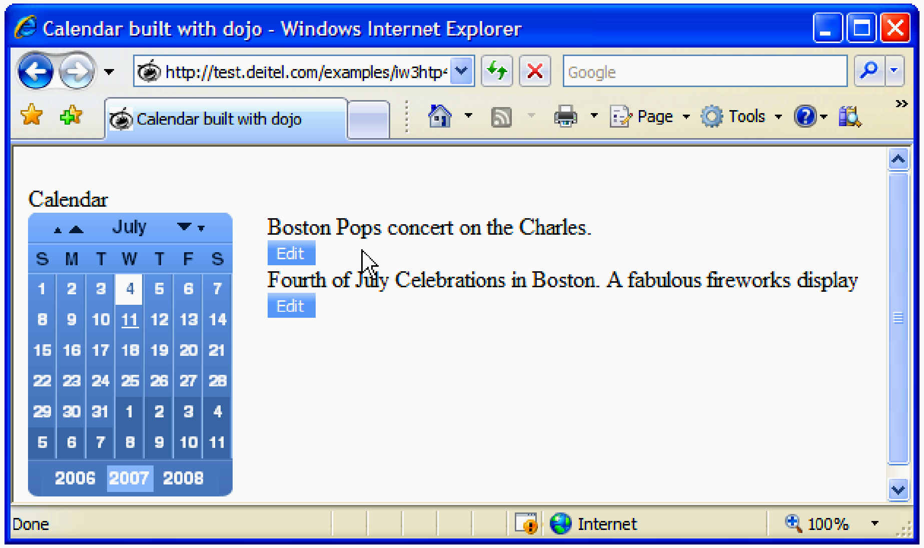
d) Application performs a partial page update, replacing the original description and the **Edit** button with a text box, **Save** button and **Cancel** button. User modifies the event description and clicks the **Save** button.



```
Boston Pops concert
on the Charles.
```

Fourth of July Celebrations in Boston. A fabulous fireworks display

e) The **Save** button's event handler uses an asynchronous request to update the server and uses the server's response to perform a partial page update, replacing the editing GUI components with the updated description and an **Edit** button.



Boston Pops concert on the Charles.

Fourth of July Celebrations in Boston. A fabulous fireworks display