1

15

Ajax-Enabled Rich Internet Applications

2

… 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

3

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

4

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.

5

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.

6

Outline

7

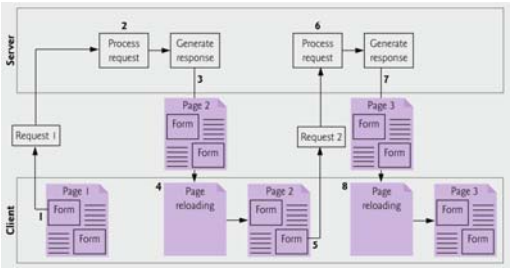


Fig. 15.1 | Classic web application reloading the page for every user interaction.

8

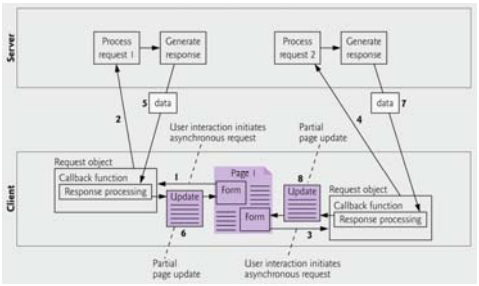


Fig. 15.2 | Ajax-enabled web application interacting with the server asynchronously.

9

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

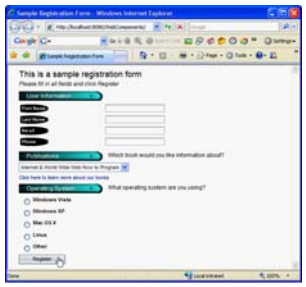


Fig. 15.3 | Classic XHTML form: User submits entire form to server, which validates the data entered (if any). Server responds indicating fields with invalid or missing data. (Part 1 of 2.)

10

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.)

11

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

12

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.

Outline

SwitchContent.html

(1 of 5)

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 15.5: SwitchContent.html -->
6  <!-- Asynchronously display content without reloading the page. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8  <head>
9     <style type="text/css">
10       .box { border: 1px solid black;
11              padding: 10px }
12    </style>
13    <title>Switch Content Asynchronously</title>
14    <script type = "text/javascript" language = "JavaScript">
15       <!--
16       var asyncRequest; // variable to hold XMLHttpRequest object
17
18       // set up and send the asynchronous request
19       function getContent( url )
20       {
21          // attempt to create the XMLHttpRequest and make the request
22          try
23          {
24             asyncRequest = new XMLHttpRequest(); // create request object
25
26             // register event handler
27             asyncRequest.onreadystatechange = stateChange;
28             asyncRequest.open( 'GET', url, true ); // prepare the request
29             asyncRequest.send( null ); // send the request
30          } // end try
```

Outline

SwitchContent.html

(2 of 5)

```
31          catch ( exception )
32          {
33             alert( 'Request failed.' );
34          } // end catch
35       } // end function getContent
36
37       // displays the response data on the page
38       function stateChange()
39       {
40          if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
41          {
42             document.getElementById( 'contentArea' ).innerHTML =
43                asyncRequest.responseText; // places text in contentArea
44          } // end if
45       } // end function stateChange
46
47       // clear the content of the box
48       function clearContent()
49       {
50          document.getElementById( 'contentArea' ).innerHTML = '';
51       } // end function clearContent
52       // -->
```

```
   </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()'/>
```

16

Outline

SwitchContent.html

(3 of 5)

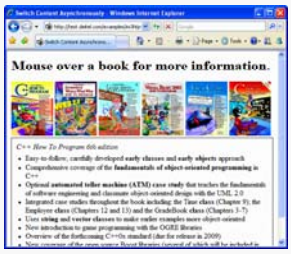

```
   <img src =
      "http://test.deitel.com/examples/iw3htp4/ajax/thumbs/chtp5.jpg"
      onmouseover = 'getContent( "chtp5.html" )'
      onmouseout = 'clearContent()'/>
   <div class = "box" id = "contentArea"> </div>
</body>
</html>
```

17

Outline

SwitchContent.html

(4 of 5)

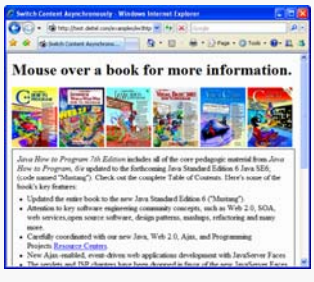


18

Outline

SwitchContent.html

(5 of 5)



19

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.

20

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.)

21

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.)

```
<?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
```

22

Outline

PullImagesOnto Page.html

(1 of 6)


```
      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' );
```

23

Outline

PullImagesOnto Page.html

(2 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
```

24

Outline

PullImagesOnto Page.html

(3 of 6)

```
85            // if there are 6 images in the row, append the row to
86            // table and start a new row
87            if ( rowCount == 6 && i + 1 < covers.length )
88            {
89               tableBody.appendChild( imageRow );
90               imageRow = document.createElement( "tr" );
91               rowCount = 0;
92            } // end if statement
93         } // end for statement
94
95         tableBody.appendChild( imageRow ); // append row to table body
96         imageTable.appendChild( tableBody ); // append body to table
97         output.appendChild( imageTable ); // append table to covers div
98      } // end if
99   } // end function processResponse
100
101  // deletes the data in the table.
102  function clearTable()
103  {
104     document.getElementById( "covers" ).innerHTML = '';
105  }// end function clearTable
```

25

Outline

PullImagesOnto Page.html

(4 of 6)


```
106   </script>
107</head>
108<body>
109   <input type = "radio" checked = "unchecked" name ="Books" value = "all"
110      onclick = 'getImages( "all.xml" )'/> All Books
111   <input type = "radio" checked = "unchecked"
112      name = "Books" value = "simply"
113      onclick = 'getImages( "simply.xml" )'/>  Simply Books
114   <input type = "radio" checked = "unchecked"
115      name = "Books" value = "howto"
116      onclick = 'getImages( "howto.xml" )'/> How to Program Books
117   <input type = "radio" checked = "unchecked"
118      name = "Books" value = "dotnet"
119      onclick = 'getImages( "dotnet.xml" )'/> .NET Books
120   <input type = "radio" checked = "unchecked"
121      name = "Books" value = "javaccpp"
122      onclick = 'getImages( "javaccpp.xml" )'/> Java, C, C++ Books
123   <input type = "radio" checked = "checked" name = "Books" value = "none"
124      onclick = 'clearTable()'/> None
125   <br/>
126   <div id = "covers"></div>
127</body>
128</html>
```

26

Outline

PullImagesOnto Page.html

(5 of 6)

27

Outline

PullImagesOnto Page.html

(6 of 6)

```
<?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
```

28

Outline

addressbook.html

(1 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
```

29

Outline

addressbook.html

(2 of 18)


```

            // 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

```

30

Outline

addressbook.html

(3 of 18)

```
86    // use the DOM to display the retrieved address book entries
87    function displayNames( data )
88    {
89       // get the placeholder element from the page
90       var listBox = document.getElementById( 'Names' );
91       listBox.innerHTML = ''; // clear the names on the page
92
93       // iterate over retrieved entries and display them on the page
94       for ( var i = 0; i < data.length; i++ )
95       {
96          // dynamically create a div element for each entry
97          // and a fieldset element to place it in
98          var entry = document.createElement( 'div' );
99          var field = document.createElement( 'fieldset' );
100         entry.onclick = handleOnClick; // set onclick event handler
101         entry.id = i; // set the id
102         entry.innerHTML = data[ i ].First + ' ' + data[ i ].Last;
103         field.appendChild( entry ); // insert entry into the field
104         listBox.appendChild( field ); // display the field
105      } // end for
106   } // end function displayAll
107
108   // event handler for entry's onclick event
109   function handleOnClick()
110   {
111      // call getAddress with the element's content as a parameter
112      getAddress( eval( 'this' ), eval( 'this.innerHTML' ) );
113   } // end function handleOnClick
114
```

31

Outline

addressbook.html

(4 of 18)


```
115   // search the address book for input
116   // and display the results on the page
117   function search( input )
118   {
119      // get the placeholder element and delete its content
120      var listBox = document.getElementById( 'Names' );
121      listBox.innerHTML = ''; // clear the display box
122
123      // if no search string is specified all the names are displayed
124      if ( input == "" ) // if no search value specified
125      {
126         showAddressBook(); // Load the entire address book
127      } // end if
128      else
129      {
130         var params = '[{"param": "input", "value": "' + input + '"}]';
131         callWebService( "search",  params , parseData );
132      } // end else
133   } // end function search
134
135   // Get address data for a specific entry
136   function getAddress( entry, name )
137   {
138      // find the address in the JSON data using the element's id
139      // and display it on the page
140      var firstLast = name.split(" "); // convert string to array
141      var requestUrl = webServiceUrl + "/getAddress?first="
142         + firstLast[ 0 ] + "&last=" + firstLast[ 1 ];
143
```

32

Outline

addressbook.html

(5 of 18)


```
144      // attempt to send an asynchronous request
145      try
146      {
147         // create request object
148         var asyncRequest = new XMLHttpRequest();
149
150         // create a callback function with 2 parameters
151         asyncRequest.onreadystatechange = function()
152         {
153            displayAddress( entry, asyncRequest );
154         }; // end anonymous function
155
156         asyncRequest.open( 'GET', requestUrl, true );
157         asyncRequest.setRequestHeader("Accept",
158         "application/json; charset=utf-8"); // set response datatype
159         asyncRequest.send(); // send request
160      } // end try
161      catch ( exception )
162      {
163         alert ( 'Request Failed.' );
164      } // end catch
165   } // end function getAddress
166
```

33

Outline

addressbook.html

(6 of 18)

```
167    // clear the entry's data.
168    function displayAddress( entry, asyncRequest )
169    {
170       // if request has completed successfully, process the response
171       if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
172       {
173          // convert the JSON string to an object
174          var data = asyncRequest.responseText.parseJSON();
175          var name = entry.innerHTML // save the name string
176          entry.innerHTML = name + '<br/>' + data.Street +
177             '<br/>' + data.City + ', ' + data.State
178             + ', ' + data.Zip + '<br/>' + data.Telephone;
179
180          // clicking on the entry removes the address
181          entry.onclick = function()
182          {
183             clearField( entry, name );
184          }; // end anonymous function
185
186       } // end if
187    } // end function displayAddress
188
```


Outline

addressbook.html

(7 of 18)


```
189    // clear the entry's data
190    function clearField( entry, name )
191    {
192       entry.innerHTML = name; // set the entry to display only the name
193       entry.onclick = function() // set onclick event
194       {
195          getAddress( entry, name ); // retrieve address and display it
196       }; // end function
197    } // end function clearField
198
199    // display the form that allows the user to enter more data
200    function addEntry()
201    {
202       document.getElementById( 'addressBook' ).style.display = 'none';
203       document.getElementById( 'addEntry' ).style.display = 'block';
204    } // end function addEntry
205
206    // send the zip code to be validated and to generate city and state
207    function validateZip( zip )
208    {
209       // build parameter array
210       var params = '[{"param": "zip", "value": "' + zip + '"}]';
211       callWebService ( "validateZip", params, showCityState );
212    } // end function validateZip
213
```


Outline

addressbook.html

(8 of 18)


```
214    // get city and state that were generated using the zip code
215    // and display them on the page
216    function showCityState( asyncRequest )
217    {
218       // display message while request is being processed
219       document.getElementById( 'validateZip' ).
220          innerHTML = "Checking zip...";
221
222       // if request has completed successfully, process the response
223       if ( asyncRequest.readyState == 4 )
224       {
225          if ( asyncRequest.status == 200 )
226          {
227             // convert the JSON string to an object
228             var data = asyncRequest.responseText.parseJSON();
229
230             // update zip code validity tracker and show city and state
231             if ( data.Validity == 'Valid' )
232             {
233                zipValid = true; // update validity tracker
234
235                // display city and state
236                document.getElementById( 'validateZip' ).innerHTML = '';
237                document.getElementById( 'city' ).innerHTML = data.City;
238                document.getElementById( 'state' ).
239                   innerHTML = data.State;
240             } // end if
```


Outline

addressbook.html

(9 of 18)

```
241             else
242             {
243                zipValid = false; // update validity tracker
244                document.getElementById( 'validateZip' ).
245                   innerHTML = data.ErrorText; // display the error
246
247                // clear city and state values if they exist
248                document.getElementById( 'city' ).innerHTML = '';
249                document.getElementById( 'state' ).innerHTML = '';
250             } // end else
251          } // end if
252          else if ( asyncRequest.status == 500 )
253          {
254             document.getElementById( 'validateZip' ).
255                innerHTML = 'Zip validation service not available';
256          } // end else if
257       } // end if
258    } // end function showCityState
259
260    // send the telephone number to the server to validate format
261    function validatePhone( phone )
262    {
263       var params = '[{ "param": "tel", "value": "' + phone + '"}]';
264       callWebService( "validateTel", params, showPhoneError );
265    } // end function validatePhone
266
```

Outline

addressbook.html

(10 of 18)


```
267    // show whether the telephone number has correct format
268    function showPhoneError( asyncRequest )
269    {
270       // if request has completed successfully, process the response
271       if ( asyncRequest.readyState == 4 && asyncRequest.status == 200 )
272       {
273          // convert the JSON string to an object
274          var data = asyncRequest.responseText.parseJSON();
275
276          if ( data.ErrorText != "Valid Telephone Format" )
277          {
278             phoneValid = false; // update validity tracker
279          } // end if
280          else
281          {
282             phoneValid = true; // update validity tracker
283          } // end else
284
285          document.getElementById( 'validatePhone' ).
286             innerHTML = data.ErrorText; // display the error
287       } // end if
288    } // end function showPhoneError
289
290    // enter the user's data into the database
```

Outline

addressbook.html

(11 of 18)


```
291    function saveForm()
292    {
293       // retrieve the data from the form
294       var first = document.getElementById( 'first' ).value;
295       var last = document.getElementById( 'last' ).value;
296       var street = document.getElementById( 'street' ).value;
297       var city = document.getElementById( 'city' ).innerHTML;
298       var state = document.getElementById( 'state' ).innerHTML;
299       var zip = document.getElementById( 'zip' ).value;
300       var phone = document.getElementById( 'phone' ).value;
301
302       // check if data is valid
303       if ( !zipValid || !phoneValid  )
304       {
305          // display error message
306          document.getElementById( 'success' ).innerHTML =
307             'Invalid data entered. Check form for more information';
308       } // end if
309       else if ( ( first == "" ) || ( last == "" ) )
310       {
311          // display error message
312          document.getElementById( 'success').innerHTML =
313             'First Name and Last Name must have a value.';
314       } // end if
```

Outline

addressbook.html

(12 of 18)

```
315         else
316         {
317            // hide the form and show the addressbook
318            document.getElementById( 'addEntry' )
319               .style.display = 'none';
320            document.getElementById( 'addressBook' ).
321               style.display = 'block';
322
323            // build the parameter to include in the web service URL
324            params = '[{"param": "first", "value": "' + first +
325               '"},{"param": "last", "value": "' + last +
326               '"},{"param": "street", "value": "' + street +
327               '"},{"param": "city", "value": "' + city +
328               '"},{"param": "state", "value": "' + state +
329               '"},{"param": "zip", "value": "' + zip +
330               '"},{"param": "tel", "value": "' + phone + '"}]';
331
332            // call the web service to insert data into the database
333            callWebService( "addEntry", params, parseData );
334         } // end else
335      } // end function saveForm
336      //-->
```


Outline

addressbook.html

(13 of 18)


```
337   </script>
338</head>
339<body onload = "showAddressBook()">
340   <div>
341      <input type = "button" value = "Address Book"
342         onclick = "showAddressBook()"/>
343      <input type = "button" value = "Add an Entry"
344         onclick = "addEntry()"/>
345   </div>
346   <div id = "addressBook" style = "display : block;">
347      Search By Last Name:
348      <input onkeyup = "search( this.value )"/>
349      <br/>
350      <div id = "Names">
351      </div>
352   </div>
353   <div id = "addEntry" style = "display : none">
354      First Name: <input id = 'first'/>
355      <br/>
356      Last Name: <input id = 'last'/>
357      <br/>
358      <strong> Address: </strong>
359      <br/>
360      Street: <input id = 'street'/>
361      <br/>
362      City: <span id = "city" class = "validator"></span>
363      <br/>
364      State: <span id = "state" class = "validator"></span>
365      <br/>
366      Zip: <input id = 'zip' onblur = 'validateZip( this.value )'/>
```


Outline

addressbook.html

(14 of 18)


```
367      <span id = "validateZip" class = "validator">
368      </span>
369      <br/>
370      Telephone:<input id = 'phone'
371         onblur = 'validatePhone( this.value )'/>
372      <span id = "validatePhone" class = "validator">
373      </span>
374      <br/>
375      <input type = "button" value = "Submit"
376         onclick = "saveForm()" />
377      <br/>
378      <div id = "success" class = "validator">
379      </div>
380   </div>
381</body>
382</html>
```

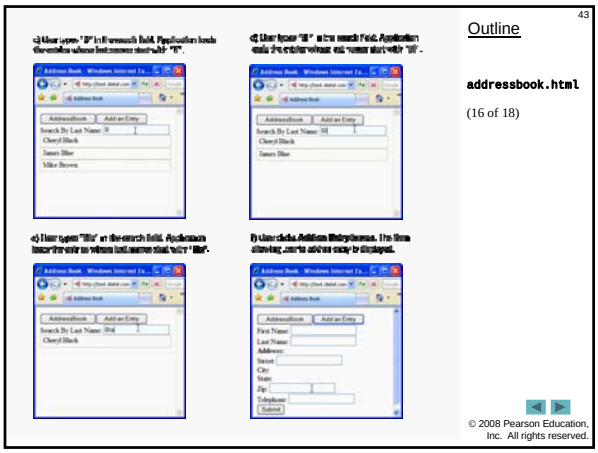
a) Page is loaded. All the entries are displayed.

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

Outline

addressbook.html

(15 of 18)

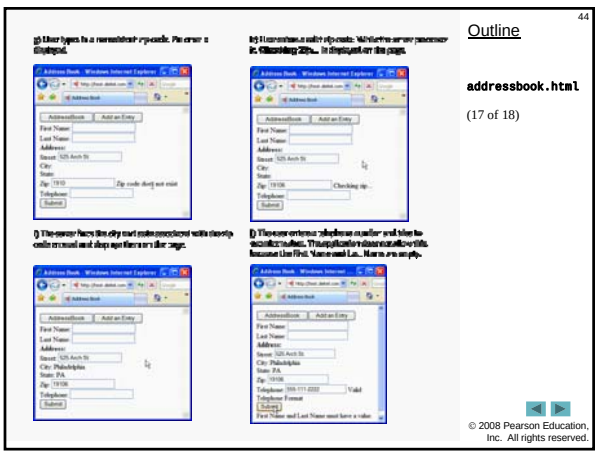


43

Outline

addressbook.html

(16 of 18)

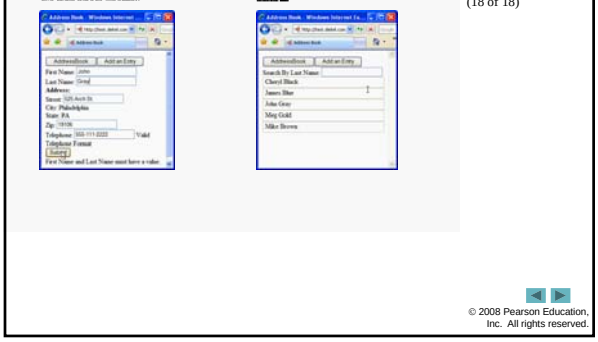


44

Outline

addressbook.html

(17 of 18)



45

Outline

addressbook.html

(18 of 18)

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


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. 15.11 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

```


Outline

Calendar.html

(1 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
```


Outline

Calendar.html

(2 of 11)

```
58          else
59             requestUrl = requestUrl + "&" + params[ i ].param +
60                "=" + params[ i ].value; // add other parameters to url
61       } // end for
62
63       // call asynchronous request using dojo.io.bind
64       dojo.io.bind( { url: requestUrl, handler: callback,
65          accept: "application/json; charset=utf-8" } );
66    } // end function callWebService
67
68    // display the list of scheduled events on the page
69    function displayItems( type, data, event )
70    {
71       if ( type == 'error' ) // if the request has failed
72       {
73          alert( 'Could not retrieve the event' ); // display error
74       } // end if
75       else
76       {
77          var placeholder = dojo.byId( "itemList" ); // get placeholder
78          placeholder.innerHTML = ''; // clear placeholder
79          var items = data.parseJSON(); // parse server data
80
81          // check whether there are events;
82          // if none then display message
83          if ( items == "" )
84          {
85             placeholder.innerHTML = 'No events for this date.';
86          }
87
```


Outline

Calendar.html

(3 of 11)


```
88          for ( var  i = 0; i < items.length; i++ )
89          {
90             // initialize item's container
91             var item = document.createElement( "div" );
92             item.id = items[ i ].id; // set DOM id to database id
93
94             // obtain and paste the item's description
95             var text = document.createElement( "div" );
96             text.innerHTML =  items[i].description;
97             text.id = 'description' + item.id;
98             dojo.dom.insertAtIndex( text, item, 0 );
99
100            // create and insert the placeholder for the edit button
101            var buttonPlaceHolder = document.createElement( "div" );
102            dojo.dom.insertAtIndex( buttonPlaceHolder, item, 1 );
103
104            // create the edit button and paste it into the container
105            var editButton = dojo.widget.
106               createWidget( "Button", {}, buttonPlaceHolder );
107            editButton.setCaption( "Edit" );
108            dojo.event.connect(
109               editButton, 'buttonClick', handleEdit );
110
111            // insert item container in the list of items container
112            dojo.dom.insertAtIndex( item, placeholder, i );
113         } // end for
114      } // end else
115   } // end function displayItems
116
```


Outline

Calendar.html

(4 of 11)


```
117   // send the asynchronous request to get content for editing and
118   // run the edit-in-place UI
119   function handleEdit( event )
120   {
121      var id = event.currentTarget.parentNode.id; // retrieve id
122      var params = '[{ "param":"id", "value":"' + id +  '"}]';
123      callWebService( 'getItemById', params, displayForEdit );
124   } // end function handleEdit
125
126   // set up the interface for editing an item
127   function displayForEdit(type, data, event)
128   {
129      if ( type == 'error' ) // if the request has failed
130      {
131         alert( 'Could not retrieve the event' ); // display error
132      }
133      else
134      {
135         var item = data.parseJSON(); // parse the item
136         var id = item.id; // set the id
137
138         // create div elements to insert content
139         var editElement = document.createElement( 'div' );
140         var buttonElement = document.createElement( 'div' );
141
142         // hide the unedited content
143         var oldItem = dojo.byId( id ); // get the original element
144         oldItem.id = 'old' + oldItem.id; // change element's id
145         oldItem.style.display = 'none'; // hide old element
146         editElement.id = id; // change the "edit" container's id
```


Outline

Calendar.html

(5 of 11)

52

Outline

Calendar.html

(6 of 11)

```
147
148         // create a textbox and insert it on the page
149         var editArea = document.createElement( 'textarea' );
150         editArea.id = 'edit' + id; // set textbox id
151         editArea.innerHTML = item.description; // insert description
152         dojo.dom.insertAtIndex( editArea, editElement, 0 );
153
154         // create button placeholders and insert on the page
155         // these will be transformed into dojo widgets
156         var saveElement = document.createElement( 'div' );
157         var cancelElement = document.createElement( 'div' );
158         dojo.dom.insertAtIndex( saveElement, buttonElement, 0 );
159         dojo.dom.insertAtIndex( cancelElement, buttonElement, 1 );
160         dojo.dom.insertAtIndex( buttonElement, editElement, 1 );
161
162         // create "save" and "cancel" buttons
163         var saveButton =
164            dojo.widget.createWidget( "Button", {}, saveElement );
165         var cancelButton =
166            dojo.widget.createWidget( "Button", {}, cancelElement );
167         saveButton.setCaption( "Save" ); // set saveButton label
168         cancelButton.setCaption( "Cancel" ); // set cancelButton text
169
170         // set up the event handlers for cancel and save buttons
171         dojo.event.connect( saveButton, 'buttonClick', handleSave );
172         dojo.event.connect(
173            cancelButton, 'buttonClick', handleCancel );
```

53

Outline

Calendar.html

(7 of 11)

```
174
175         // paste the edit UI on the page
176         dojo.dom.insertAfter( editElement, oldItem );
177      } // end else
178   } // end function displayForEdit
179
180   // sends the changed content to the server to be saved
181   function handleSave( event )
182   {
183      // grab user entered data
184      var id = event.currentTarget.parentNode.parentNode.id;
185      var descr = dojo.byId( 'edit' + id ).value;
186
187      // build parameter string and call the web service
188      var params = '[{ "param":"id", "value":"' + id +
189         '"}, {"param": "descr", "value":"' + descr + '"}]';
190      callWebService( 'Save', params, displayEdited );
191   } // end function handleSave
192
193   // restores the original content of the item
194   function handleCancel( event )
195   {
196      var voidEdit = event.currentTarget.parentNode.parentNode;
197      var id = voidEdit.id; // retrieve the id of the item
198      dojo.dom.removeNode( voidEdit, true ); // remove the edit UI
199      var old = dojo.byId( 'old' + id ); // retrieve pre-edit version
200      old.style.display = 'block'; // show pre-edit version
201      old.id = id; // reset the id
202   } // end function handleCancel
203
```

54

Outline

Calendar.html

(8 of 11)

```
204   // displays the updated event information after an edit is saved
205   function displayEdited( type, data, event )
206   {
207      if ( type == 'error' )
208      {
209         alert( 'Could not retrieve the event' );
210      }
211      else
212      {
213         editedItem = data.parseJSON(); // obtain updated description
214         var id = editedItem.id; // obtain the id
215         var editElement = dojo.byId( id ); // get the edit UI
216         dojo.dom.removeNode( editElement, true ); // delete edit UI
217         var old = dojo.byId( 'old' + id ); // get item container
218
219         // get pre-edit element and update its description
220         var oldText = dojo.byId( 'description' + id );
221         oldText.innerHTML = editedItem.description;
222
223         old.id = id; // reset id
224         old.style.display = 'block'; // show the updated item
225      } // end else
226   } // end function displayEdited
227
228   // when the page is loaded, set up the calendar event handler
229   dojo.addOnLoad( connectEventHandler );
230   // -->
```

```
   </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

(9 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.

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

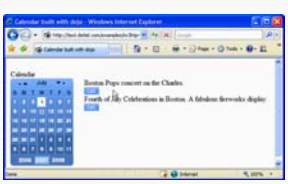
Outline

Calendar.html

(10 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.

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.



Outline

Calendar.html

(11 of 11)