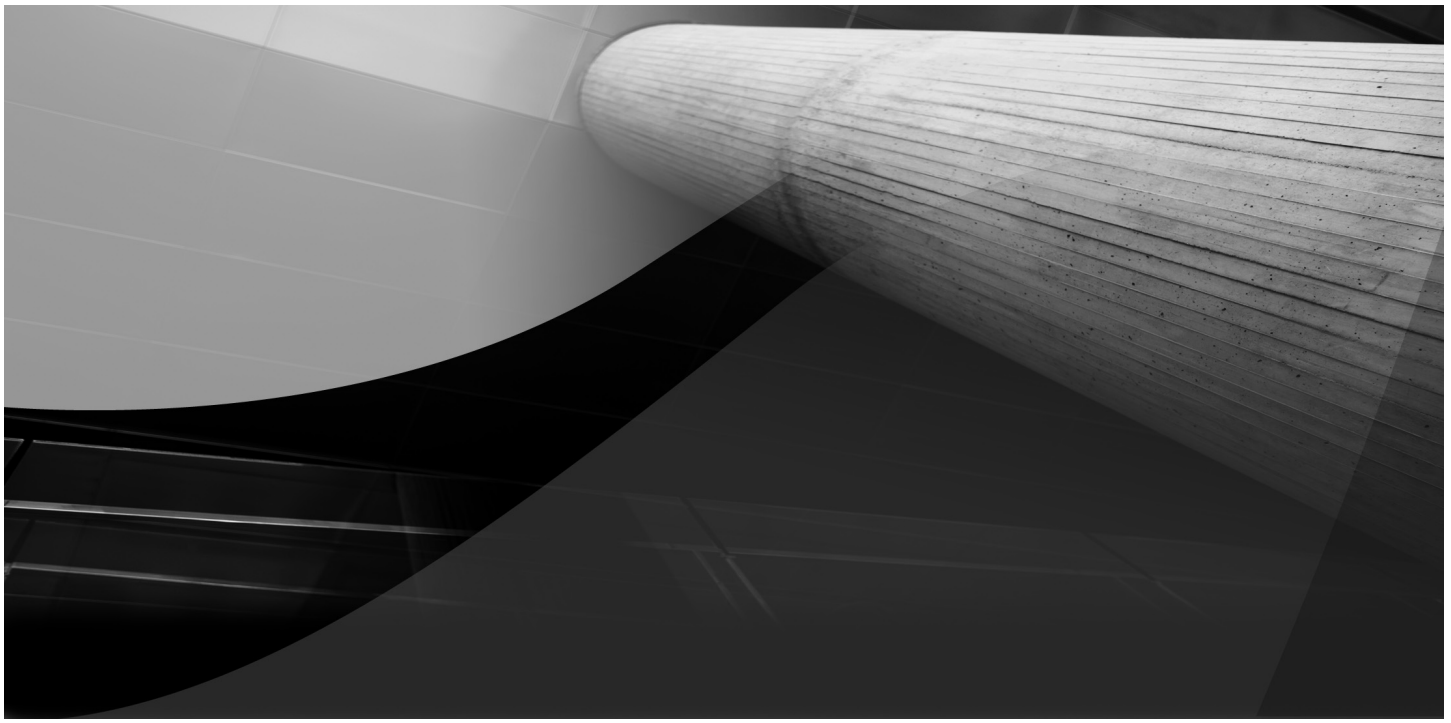

CHAPTER 11

Creating User-Defined Mashups

s powerful computers have become more common, users have begun to see them as information aggregation tools. This is a direct outgrowth of the web experience. Web sites are a version of data aggregation that allows users to select the information to be viewed. Today's browsers also allow web RSS feeds to show up as links in a page or as bookmarks. E-mail has long since been included in the browser. All of these, as well as AJAX applications, lead a user to believe that data aggregation is the purpose of the browser and therefore the computer itself.

As these users demand more and more application flexibility, web applications must adapt. Using *mashups*, multiple portions of separate pages displayed in one page, is one way of accomplishing this. One description of what a mashup is can be found at http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29.

After reading this chapter you will be able to create pages that let the user combine data from various remote web sites or applications into one AJAX-driven web application. They will be able to define what portions of the pages to display and where these portions should be displayed on the page.

The items covered in this chapter are

- Using JavaScript to define and display multiple portions of web pages in a single page
- Using JSON for data storage and retrieval
- The mashup library API

These dynamic mashups allow the user to customize your application with little effort on both their part and yours.

Creating a Simple Mashup Page

Home health care nurses do not have as much control of their patients' environment as do nurses in a hospital. Because of this they need to monitor the air temperature and quality in the area where the patient resides. While there are many sites that can supply a portion of this information for any given location, there are few or none that supply it all. To keep on top of this data and be proactive, a nurse would need to regularly check several sites throughout the day. With the extreme time constraints that these people work under and the potential loss of health and life if this type of information is ignored, a simple way for them to select their preferred information sources and display them is needed. A web page mashup is just the trick.

A web page mashup consists of bits of other complete web pages included within one single page. In its simplest form, such a mashup is defined by the programmer or engineer and doesn't allow the user to add new pages. The mashupExample.html file,

downloadable from www.OraclePress.com and seen in Figure 11-1, is similar to such a simple mashup and displays air quality information from the Salt Lake City, Utah area.

Mashup frames consist of the page display and the ability to resize the display, select the portion of the source page to be displayed, move the display, and remove the display. The mashupExample.html page also includes a button that allows the user to temporarily add new mashup frames. All changes made to this page are temporary because they are not being sent to a server for storage. Storing this definition information to Oracle Database is covered later in this chapter.

Since these mashup frames are accessing the source pages via the web, they always display information from the current page on the remote server. Thus, when the source page changes, the display in the mashup changes as well. This allows the data to always be up to date. Because of this, one factor to take into account when selecting a source page for a mashup frame is the stability of the layout of the page. Since only a selected region within the source page is displayed, if the source page's layout changes, the mashup frame will display the new content for the old region.

The API, seen in Table 11-1, used to create mashups consists of three functions and is dependent on the drag-and-drop library covered in Chapter 6. These three

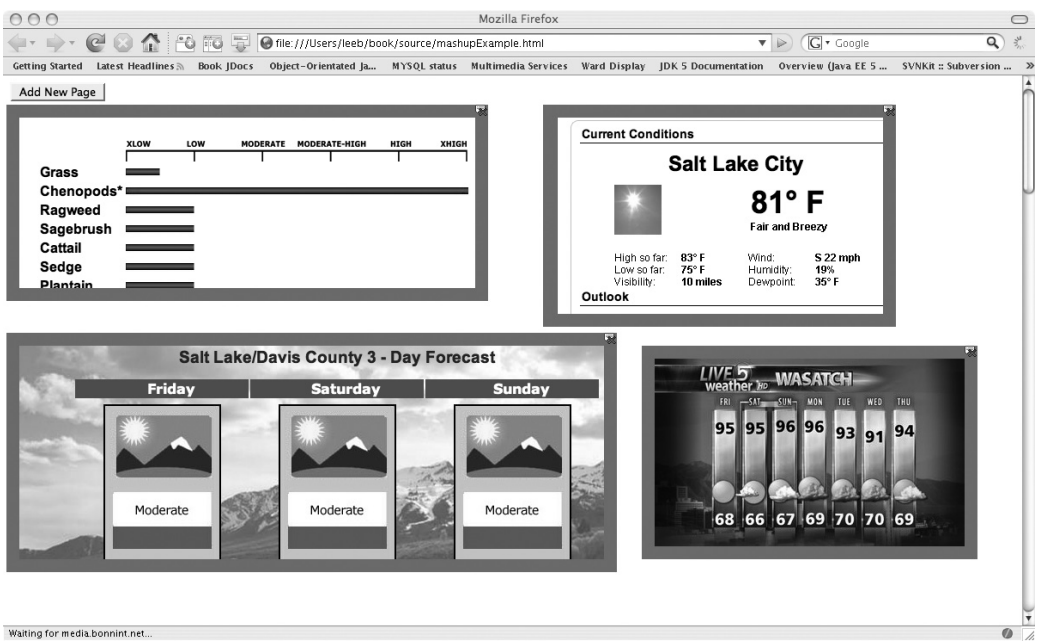

FIGURE 11-1. *A simple mashup page in Firefox on OS X*

Function	Description
MashFrame (a URL, parentID, xLoc, yLoc, width, height, scrollDown, scrollRight)	This function is the constructor for a mashup frame. It has two required parameters and six optional parameters. Required: ■ **a URL** The URL of the source page. This need not contain the "http://" portion but can if desired. All URLs are assumed to be HTTP and not FTP or some other protocol. ■ **parentID** The id of the HTML element that is to be the container for the mashup frame. This is usually an HTML div. Optional: ■ **xLoc** The horizontal offset within the containing HTML element for the left side of the mashup frame. The default value is 0 pixels. ■ **yLoc** The vertical offset within the containing HTML element for the top of the mashup frame. The default value is 0 pixels. ■ **width** The width of the mashup frame to be displayed in pixels from the leftmost to the rightmost pixel. The default value is 300 pixels. ■ **height** The height of the mashup frame to be displayed in pixels from the topmost to the bottommost pixel. The default value is 294 pixels. ■ **scrollDown** The vertical offset in pixels of the viewable region as if the user had scrolled down the source page. The default is 0 pixels. ■ **scrollRight** The horizontal offset in pixels of the viewable region as if the user had scrolled right in the source page. The default is 0 pixels.
requestNewMashFrame (parentID)	This function is a wrapper for the MashFrame constructor that uses the default values for all of the optional parameters. When executed, this function prompts the user for the URL of the source page. It has one required parameter. ■ **parentID** The id of the HTML element that is to be the container for the mashup frame. This is usually an HTML div.
getMashupDescriptor()	This function returns an array of mashup frame descriptors. Each descriptor consists of all of the current values for a single mash frame. The returned array is used to store the current state of all of the mashup frames to the server.

TABLE 11-1. *The Mashup API*

functions allow the programmer to define mashup frames and retrieve a description of all of the frames' current state. The mashup library, found in mashup.js and downloadable from www.OraclePress.com, has been written to support the latest versions of Firefox and Safari on OS X, as well as Firefox and IE on Windows. Further work may be needed to support Firefox on the various flavors of Linux.

The mashupExample.html file uses two of the API functions listed in Table 11-1. It consists of four mashup frames positioned and sized differently to display the current temperature and air quality information for the Salt Lake City, Utah area.

```
<html>
<head>
   <link rel="stylesheet" type="text/css" href="mashup.css" />
   <script src="util.js" type="text/javascript"></script>
   <script src="mashup.js" type="text/javascript"></script>
   <script src="JSON_Util.js" type="text/javascript"></script>

   <script>
    function init(){
        new mashFrame('www.intermountainallergy.com/pollen.html',
'displayDiv',0,30,550,200,300,60);
        new mashFrame('http://www.airquality.utah.gov/slc.html',
'displayDiv',0,295,700,250,210,200);
        new mashFrame('www.ksl.com/index.php?nid=88',
'displayDiv',625,30,400,230,850,0);
        new mashFrame('www.ksl.com/index.php?nid=88',
'displayDiv',740,310,380,220,550,40);
    }
   </script>
</head>
<body id='mainBody' onload='init()'>
<input type = 'button' value='Add New Page'
onclick='requestNewMashFrame("displayDiv")' />
<div id='displayDiv' style='width: 1000px; height: 3000px;
top: 50px;' >

</div>

</body>
</html></div>
</body>
</html>
```

Notice that the displayDiv element is set to a large width and height. This was necessary because when a mashup frame is being dragged around or resized, sometimes the mouse will leave the grey bar used to drag or resize the mashup frame. This is common in browsers due to the time required to interpret and execute

the JavaScript code and then render the changes. To overcome this, the mashup library, via the drag-and-drop library util.js, adds a mouse listener to the parent that causes the mashup frame to "catch up" to the mouse.

Occasionally this same problem happens within the display area of the mashup frame as well. On all browsers except IE the mashup frame will again catch up with the mouse. When moving or resizing a mashup frame in IE, care must be taken to move the mouse slowly enough that it will not enter the display area of the frame. The reason for this is that in IE many active components such as links always have the highest z-order, regardless of the z-order declared for them or their parents, and therefore the catch-up code could not be implemented.

Since the mashup library, mashup.js, uses a clear, overlaying div known as a *glass pane* to cover the display area, it can detect a straying mouse move and make the mashup frame catch up. In IE this glass pane has been removed since the active components of the source page would all be above it anyway. Because of this z-order implementation in IE, the individual active components would capture the mouse motion and the mashup frame would not be notified to catch up to the mouse. For consistency, the glass pane was removed. Other limitations exist in the mashup library as well.

If the source page includes Flash or other embedded interpreters, the resultant display of the movie or other content can appear even though it is outside the viewable area. This issue appears to be more prevalent on Windows, for both Firefox and IE, than on OS X and has not been tested on Linux. It appears to be more site dependent than browser dependent and as such means that careful consideration needs to be given to source page selection.

Another limitation is JavaScript menus. Some sites have menus that automatically pop up as the page is loaded. These menus also can sometimes appear even though they are outside the display region of the mashup frame.

There is also a difference in functionality between how the mashup library works on Windows and other operating systems. Figure 11-1 shows a simple example in Firefox on OS X. Notice that there are no scrollbars within the mashup frames. Here, dragging it within the mashup frame changes the region of the source page being displayed. When this drag-and-drop approach to source page region display was used in Firefox and IE on Windows, both browsers would misrender the display area and leave behind visual artifacts outside the display region. This problem went away if scrollbars were used instead, as seen in Figure 11-2.

These visual artifacts did not appear in Firefox or Safari on OS X, so it appears that the issue is within the MFC Windows library used for visual display in both of these Windows browsers.

It is hoped that as time goes by, the mashup library will become stronger and more sophisticated so that these issues can be overcome.

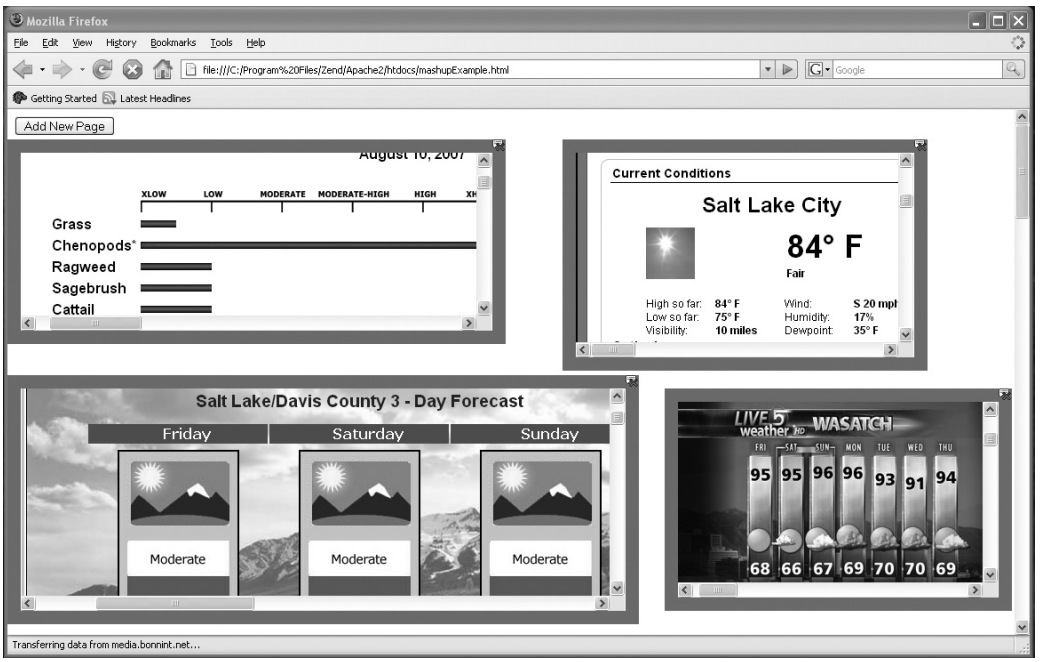

FIGURE 11-2. *The simple mashup page displayed in Firefox on Windows*

Embedding Mashups in the Medical Data System

As stated at the beginning of the previous section, the use of mashups can be of great worth to home health care nurses. As such it is important to make the mashup program simple for them to use, and it needs to remember the choices they have made. If the same user interface components are used as in the other portions of the medical data system, the user can easily use the functionality. By using the same design approach as in the other chapters in this book, the programmer can more readily include the functionality. Figure 11-3 shows mashup frames included in the main page.

Chapter 10 covers how the client-side Session object is stored in and retrieved from Oracle using AJAX. Since the mashup choices made by the user need to be stored, they can be added to the Session object, as was the initial subpage choice in that chapter. The login process from Chapter 10 will then retrieve these choices and they will be available from within the Session object when the user chooses to display the mashup view.

FIGURE 11-3. *The medical data system after clicking Show Mashup*

In order to piggy-back on the code to store the Session object, a BCO needs to be created as described in Chapter 3. The saveMashupBCO is found in the CO.js file, also downloadable from www.OraclePress.com.

```
function saveMashupBCO(){
    //create or replace the attribute with a new one
    session.addAttribute('mashupDesc', getMashupDescriptor());
    //post the session string to the server using no VCO
    theSAO.makeCall('POST', null, 'Text', true, '',
'cmd=store&sessDef='+session.toJSONString());
}
```

The saveMashupBCO inserts the descriptions of all of the user-defined mashup frames by adding the results of the **getMashupDescriptor** API function as an attribute of the Session object. Then piggy-backing on the client session storage functionality created on the server in Chapter 10, it makes the same HTTP POST call as seen there. This will cause the descriptions to be stored in the field in the database that contains the session as a JSON string. To find out more about JSON see the last section of Chapter 10.

As seen in previous chapters, when the user selects to see the mashup view, a BCO and a matching VCO need to be accessed. The mashupBCO, as seen in the following code and in CO.js, differs from most of the JavaScript BCOs seen so far in that it does not contact the server for data. When the user logged in, the Session object that was instantiated already included all of the mashup description information needed if the user had already stored it.

```
function mashupBCO(){
    //no connection to the server is required since the
    //client session information was retrieved at login
    var mashupDescriptorArray = session.getAttribute('mashupDesc');
    var aVCO = new mashupVCO();
    aVCO.notify(mashupDescriptorArray);
}
```

Because of this, the BCO needs only to retrieve these descriptions from the Session object and call its matching VCO directly, passing the description data as a parameter.

Both of these BCOs are very simple. The mashupVCO, found in CO.js, is very simple as well because of the use of the mashup library.

```
function mashupVCO(){
    this.notify = function(data){
        var displayString  = "<div>";
        displayString += "<input type='button' value='Add Another Page'
onclick='requestNewMashFrame(\"mashupContainer\")' />";
        displayString += "<input type='button' value='Save Mashup
Changes' onclick='saveMashupBCO()' />";
        displayString += "<div id='mashupContainer' style='height:
4000px; width: 2000px;'></div></div>";
        document.getElementById('content').innerHTML = displayString;

        MashFrame.mashCount = 0;
        MashFrame.mashArray = new Array();
        if(data != null){
            var numFrames = data.length;
            for(var i = 0; i < numFrames; i++){
                var aDescription = data[i];
                new MashFrame(aDescription.URL, 'mashupContainer',
aDescription.left, aDescription.top,
                              aDescription.width, aDescription.height,
aDescription.scrollDown, aDescription.scrollRight);
            }
        }
    }
}
```

The mashupVCO sets up two divs for display purposes. The first one contains buttons to add new mashup frames and save mashup descriptions to the server. The second div is the container for the mashup frames themselves. Additionally, this VCO creates a MashupFrame object for each stored description and places it in the appropriate parent.

How It Is Done

The mashup library, found in mashup.js and downloadable from www.OraclePress.com, depends on the understanding of one major concept, element manipulation using JavaScript to change CSS style attributes. In order not to reinvent the wheel, much of this CSS style manipulation is done using the drag-and-drop library covered in Chapter 6. By using that pre-existing library, found in util.js, behaviors such as manipulating the location, width, and height of mashup frames are handled using prebuilt functionality. When the client is running on OS X, the moving of the web page display within the mashup frame is also done using drag and drop, as opposed to using scrollbars when it is running under Windows.

Other than this, the only other item required to understand what is happening in the mashup library is that each mashup frame consists of two main components, an iframe used to display the requested page, and a div that holds the iframe and has its CSS overflow style set to *hidden*. In order to help you to understand how this works, a page (mashupBasics.html) has been created that doesn't use the drag-and-drop library. This page, seen in Figure 11-4, allows the user to manipulate the same CSS attributes that the drag-and-drop library does but using input fields instead.

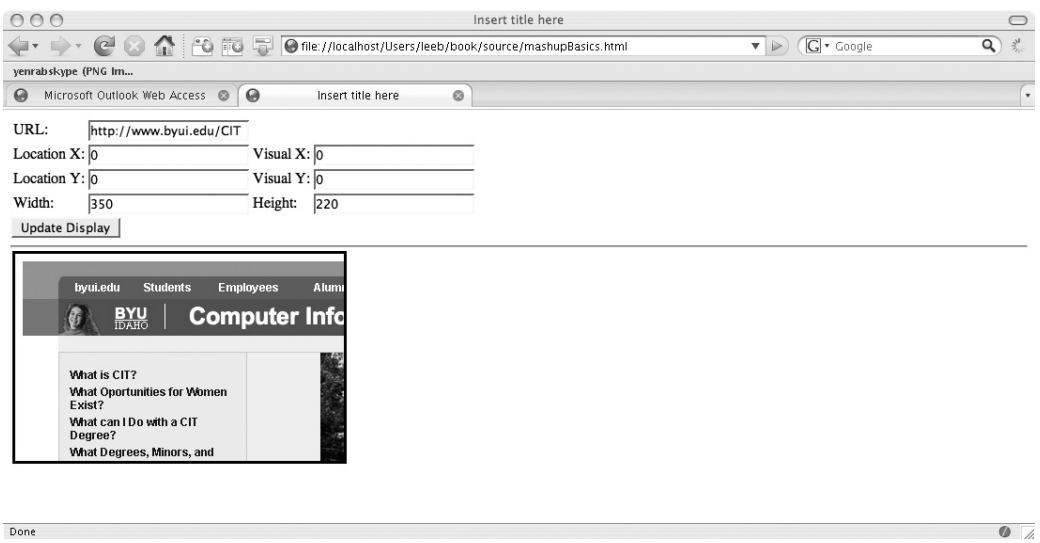

FIGURE 11-4. *The mashup basics page*

From this example of being able to manipulate the CSS style attributes directly instead of using the drag-and-drop library, you can understand the underlying simplicity of embedding mashup frames.

The mashupBasics.html file, as seen in the following code and downloadable from www.Oracle.com, consists mainly of the HTML used to define the display. The JavaScript functionality consists of two functions, **updateEmbeddedPage** and **getUpperLeftPoint**. It is important to understand that the **updateEmbeddedPage** function is not used in the mashup library, but has been created to allow the same types of manipulations that the library enables via drag and drop.

```
<html>
<head>
<title>Mashup Basics</title>
<style>
#containerDiv{
    width: 1px;
    height: 1px;
}
#maskDiv{
  width: 350px;
  height: 220px;
  border: solid;
  position: absolute;
  top: 150px;
  left: 10px;
  overflow: hidden;
  background-color: white;
}
#display{
  width: 1000px;
  height: 5000px;
  border: none;
  position: absolute;
}
</style>
<script>
function updateEmbeddedPage(){
  var basePoint = getUpperLeftPoint(document.getElementById('containerDi
v'));
  var aURL = document.getElementById('url').value || 'http://www.byui.
edu/CIT';
  var locX = document.getElementById('locX').value || 0;
  var locY = document.getElementById('locY').value || 0;
  var visX = document.getElementById('visX').value || 0;
  var visY = document.getElementById('visY').value || 0;
```

```
  var width = document.getElementById('width').value || 350;
  var height = document.getElementById('height').value || 220;
  //set the URL of the iframe
  if(document.getElementById('display').src != aURL){
    document.getElementById('display').src = aURL;
  }
  //set the location of the top and left of the
  //containing div
  var maskDiv = document.getElementById('maskDiv');
  maskDiv.style.top = (basePoint.topValue+(locY*1))+'px';
  maskDiv.style.left = (basePoint.leftValue+(locX*1))+'px';
  //set the width and height of the containing div
  maskDiv.style.width = width+'px';
  maskDiv.style.height = height+'px';
  //set the location of the top and left of the
  //iframe
  var display = document.getElementById('display');
  display.style.top = visY+'px';
  display.style.left = visX+'px';
}
//find the top and left values in
//pixels of an element in
//the page where 0,0 is the top and
//left of the page and not any other
//container of the element.
function getUpperLeftPoint(aNode){
    var aPoint = new Object()
    aPoint.leftValue = aNode.offsetLeft;
    aPoint.topValue = aNode.offsetTop;
        while((aNode = aNode.offsetParent) != null){
            aPoint.leftValue += aNode.offsetLeft;
            aPoint.topValue += aNode.offsetTop;
        }
        return aPoint;
}
</script>
<body>
<table>
  <tr>
    <td>URL: </td>
    <td><input id='url' value='http://www.byui.edu/CIT'</td>
  </tr>
  <tr>
    <td>Location X: </td>
    <td><input id='locX' value='0'/></td>
    <td>Visual X: </td>
    <td><input id='visX' value='0'/></td>
  </tr>
```

```
  <tr>
    <td>Location Y: </td>
    <td><input id='locY' value='0'/></td>
    <td>Visual Y: </td>
    <td><input id='visY' value='0'/></td>
  </tr>
  <tr>
    <td>Width: </td>
    <td><input id='width' value='350'/></td>
    <td>Height: </td>
    <td><input id='height' value='220'/></td>
  </tr>
</table>
<input type='button' value='Update Display' onclick='updateEmbeddedPag
e()';
<hr/>

<div id='containerDiv'>
  <div id='maskDiv'>
    <iframe id='display' src="http://www.byui.edu/CIT" ></iframe>
  </div>
</div>
<div style='position: absolute; top: 500px;'>
</div>
</body>
</html>
```

The **updateEmbeddedPage** function begins by retrieving all of the values from the inputs in the page and storing them in variables inside the JavaScript. If the user has not entered values into the input elements, default values are assigned to the variables instead. Once this is done, the process of manipulating the CSS styles begins.

As stated earlier, two main elements exist, a containing div called maskDiv and an iframe called *display*. The iframe's *src* attribute, which holds the location of the page it is to display, is set to the value found in the URL input element. As with any iframe, when this is done the page display is refreshed.

The top and left of the iframe *display* are set to be relative to the top, left point of the entire page. This enables it to float freely within maskDiv. Since it can float freely, it can appear to move around within maskDiv when in fact it is moving around within the page instead. In spite of this, it is still contained by maskDiv, so maskDiv's CSS overflow setting of *hidden* still applies. This causes any portion of the iframe that exists outside of maskDiv to appear to be clipped off. This appearance that the iframe is clipped off gives a mashup frame its special qualities.

Using the mashupBasics.html file to experiment with the positions and widths of its elements allows you to understand what the drag-and-drop components of the mashup library must be able to do. It also prepares you to begin looking at the code for the mashup library itself.

Summary

By leveraging the mashup library as well as the client session object and storage from Chapter 10, mashup frame inclusion is easily done in the health care application as well as any other web page or application you desire. It allows the user to aggregate additional data that they believe is needed into your application without changing the application code. By doing so, the user becomes more a partner than ever before in the production of web applications and gains some personal ownership of them without having to know code or expend even small amounts of time. Including mashups in your application will ease your load and dramatically increase the positive aspects of the user experience.