Pro Ajax and the .NET 2.0 Platform

Daniel Woolston

Apress®

Pro Ajax and the .NET 2.0 Platform

Copyright © 2006 by Daniel Woolston

ISBN-13 (pbk): 978-1-59059-670-8

ISBN-10 (pbk): 1-59059-670-6

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Matthew Moodie
Technical Reviewer: Nick McCollum
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Julie M. Smith
Copy Edit Manager: Nicole LeClerc
Copy Editor: Ami Knox
Assistant Production Director: Kari Brooks-Copony
Senior Production Editor: Laura Cheu
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: April Eddy
Indexer: Broccoli Information Management
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Source Code section.

CHAPTER 13

Ajax and Mapping

One of the more popular Ajax-enabled applications on the web is the *map-enabled utility site*. Commonly referred to as *mash-ups*, these applications are typically a combination of various technologies all rolled into one useful tool. Microsoft's Virtual Earth (VE) application is one such application that has become a central tool to many of the mash-ups on the web. At its core, the VE library provides a scrollable Ajax map that can be dropped onto a web page seamlessly, as shown in Figure 13-1.

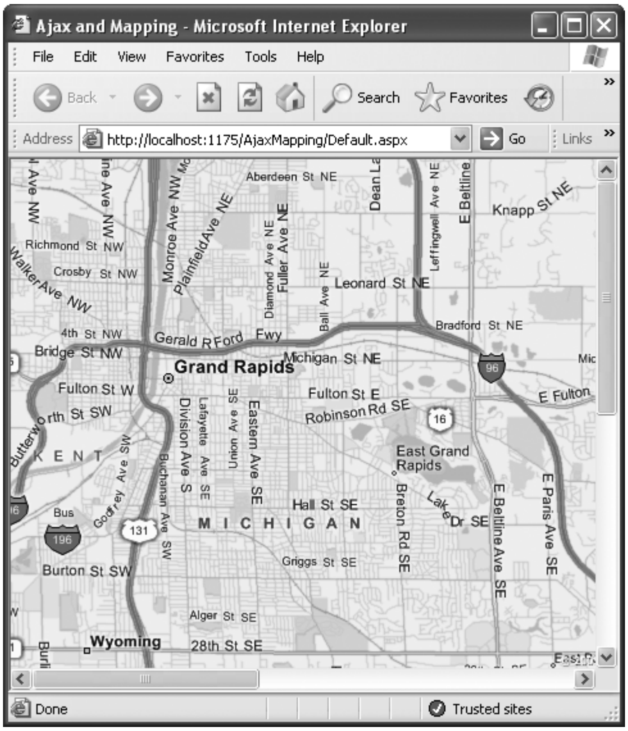

Ajax and Mapping - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address http://localhost:1175/AjaxMapping/Default.aspx
Go
Links
Grand Rapids
MICHIGAN
KENT
East Grand Rapids
Wyoming
Done
Trusted sites

Figure 13-1. *Microsoft's Virtual Earth*

Virtual Earth is much more than just a draggable map. You can insert pushpins at designated points on the map and create layovers for various user-interaction tasks. A growing number of such applications are appearing almost daily on the web. One such application that I've found to be an interesting implementation of the Virtual Earth engine is Fresh Logic Studios' Atlas mapping site (not to be confused with the Atlas engine by Microsoft); it is an excellent example of what can be done with the Virtual Earth library, as Figure 13-2 demonstrates. They've created a site that has the map as its base technology and mashed up other tools on top of the VE.

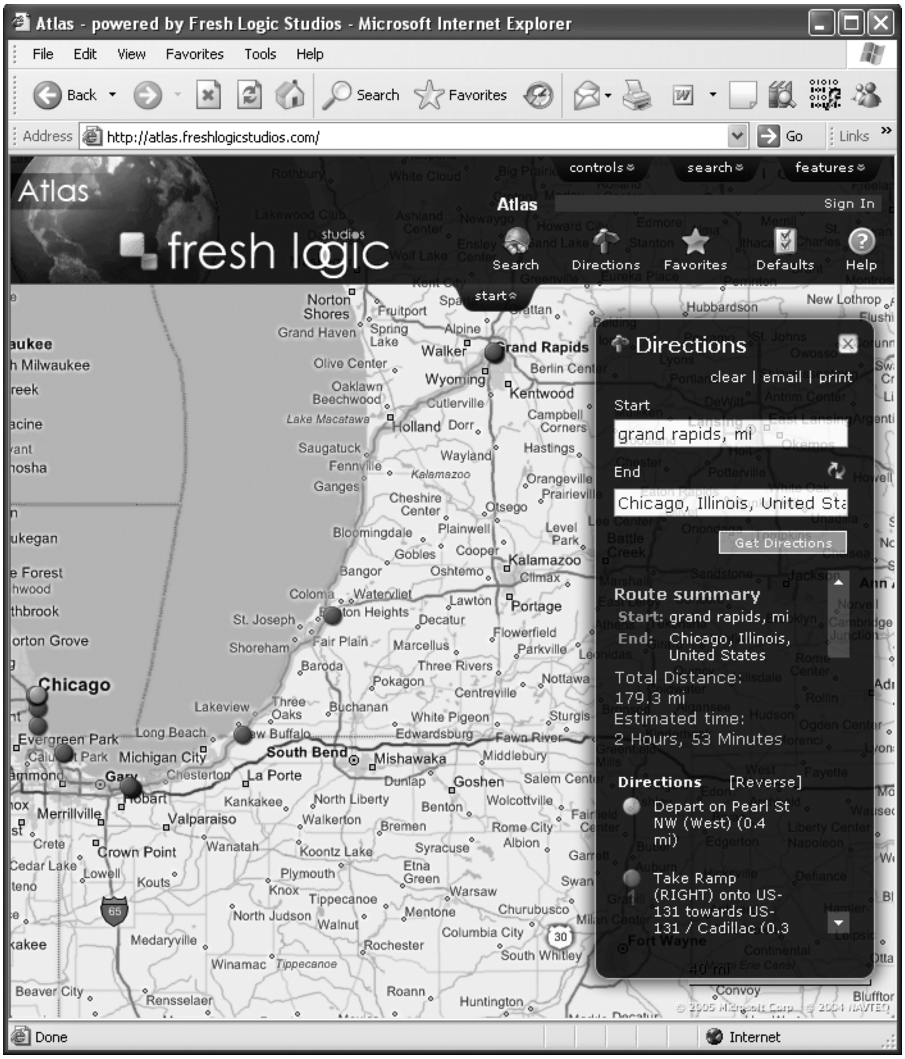

Atlas - powered by Fresh Logic Studios - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://atlas.freshlogicstudios.com/ Go Links
controls search features
Atlas Sign In
fresh logic studios
Search Directions Favorites Defaults Help
start
Directions
clear | email | print
Start
grand rapids, mi
End
Chicago, Illinois, United Sta
Get Directions
Route summary
Start: grand rapids, mi
End: Chicago, Illinois, United States
Total Distance: 179.3 mi
Estimated time: 2 Hours, 53 Minutes
Directions [Reverse]
Depart on Pearl St NW (West) (0.4 mi)
Take Ramp (RIGHT) onto US-131 towards US-131 / Cadillac (0.3
Done
Internet

Figure 13-2. *Fresh Logic Studios' Atlas site*

For this chapter, I'll show you how to create a Virtual Earth mash-up. We'll be using the standard VE library files, available for free from Microsoft by searching for "Virtual Earth SDK" on the download site at `http://www.microsoft.com/downloads`; these files are also part of the sample application for this chapter.

The site that we will be building will combine Ajax, Virtual Earth, and Yahoo's traffic RSS feed. I should also credit Peter A. Bromberg, PhD, for his awesome article and VS 2003 application that mirrors the one we will be developing in this chapter. You can find them at Peter's blog: `http://petesbloggerama.blogspot.com`.

Using VE as a base for the page, we'll dynamically add pushpins of various traffic reports to the map. Yahoo has made a traffic RSS feed available to the public that includes not only a description of the event, but also longitude and latitude locators as well. We'll discuss the feed in more detail as well as the Virtual Earth library, but for now let's take a look at what the application can do and the associated solution and source code.

Ajax Traffic

In our application, the page will display a VE map centered on a hard-coded ZIP code (more on this in a moment), and pushpins will appear for various traffic delays that are near the aforementioned ZIP code, as you see in Figure 13-3.

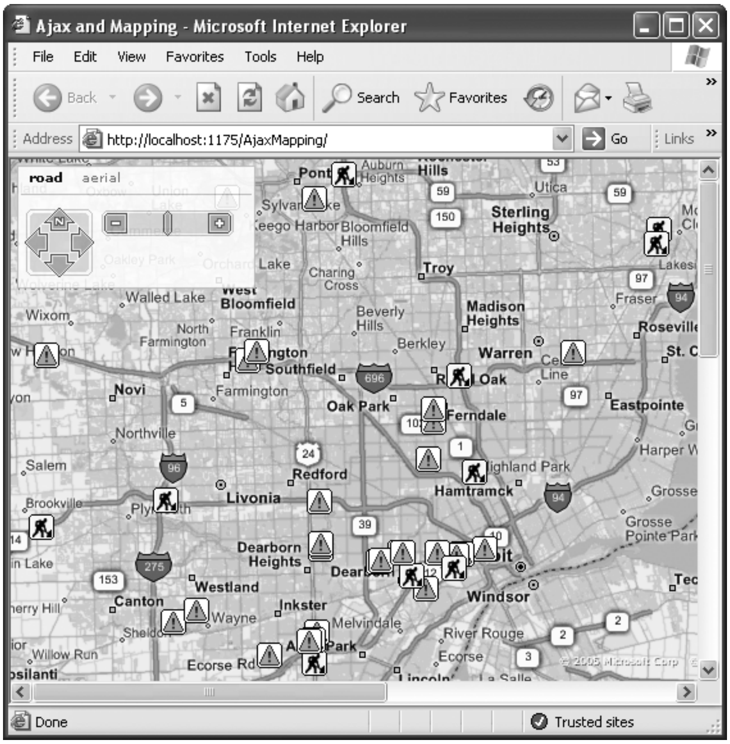
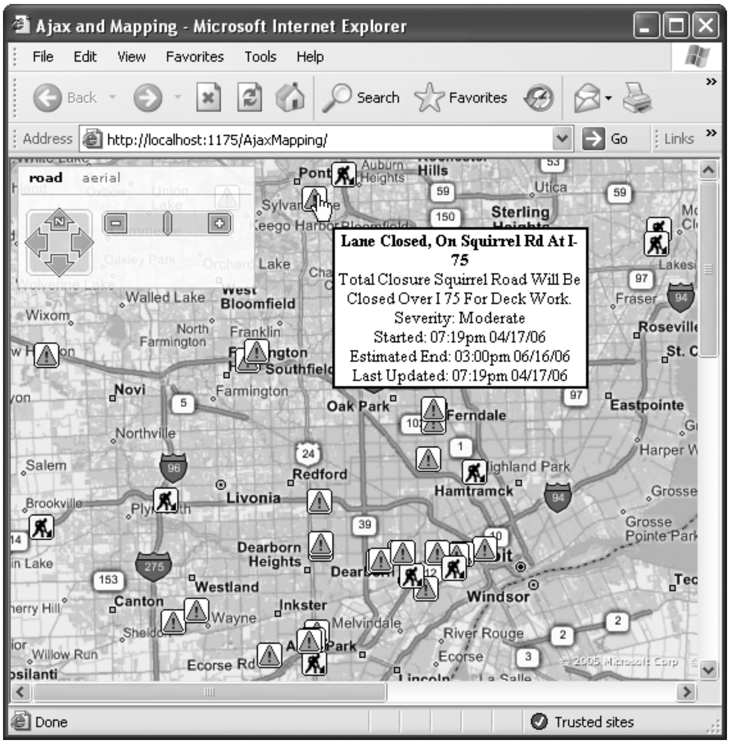


Figure 13-3. *Ajax traffic demonstration*

There are a few things worth noting concerning our application. As you can see, we have various traffic incidents in the Detroit area. You'll also notice that we have a variety of pushpins that have been applied to the map. Those are actually custom pushpins that did not come with the SDK. You'll see later how to add your own custom pushpins as well. Also notice that we have a floating toolbar that is home to our zoom controls. One other thing to note about the site is that each pushpin has a `mouseover` event that will render a pop-up containing details of the traffic report, as shown in Figure 13-4.

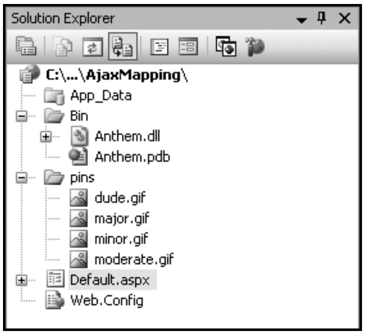

Ajax and Mapping - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:1175/AjaxMapping/
Go Links
road aerial
Done
Trusted sites

Lane Closed, On Squirrel Rd At I-75
Total Closure Squirrel Road Will Be Closed Over I 75 For Deck Work.
Severity: Moderate
Started: 07:19pm 04/17/06
Estimated End: 03:00pm 06/16/06
Last Updated: 07:19pm 04/17/06

Figure 13-4. *Traffic pop-up*

You could, of course, customize the application (and I hope that you do) to center on your own hometown with the appropriate traffic for your area.

So let's open the solution and take a look at the file list, displayed in Figure 13-5.



```
C:\...\AjaxMapping\
  App_Data
  Bin
    Anthem.dll
    Anthem.pdb
  pins
    dude.gif
    major.gif
    minor.gif
    moderate.gif
  Default.aspx
  Web.Config
```

Figure 13-5. *Ajax traffic solution*

We should first discuss the various files in brief:

- `Anthem.dll`: Ajax class library
- *Pin images*: Pushpins for the map
- `Default.aspx`: Our only web page and home to the map
- `Web.Config`: Standard `Web.Config` file

I've separated the custom pushpins out into their own directory. Each of the pushpins, illustrated in Figures 13-6 through 13-9, represents the category or severity of the traffic report.

Figure 13-6. *dude.gif—construction zone*

Figure 13-7. *minor.gif—minor traffic incident*

Figure 13-8. *moderate.gif—moderate traffic incident*

Figure 13-9. *major.gif—major traffic incident*

As I've said, you can modify these in a graphics editor of your choice, and they'll still work as planned.

Default.aspx

Now that we've taken a brief look at the overall structure of the application, let's take a look at the HTML and C# code for `Default.aspx`.

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Ajax and Mapping</title>
```

```
<link href="http://dev.virtualearth.net/standard/v2/MapControl.css"
      type="text/css" rel="stylesheet" />
<script src="http://dev.virtualearth.net/standard/v2/MapControl.js"
        type="text/javascript"></script>

<style type="text/css" media="screen">
  #popup
  {
  POSITION:absolute;VISIBILITY:hidden;Z-INDEX:199;
  }

  .pinConstruction
  {
  cursor:pointer;text-decoration:none;
  z-index:5;
  }

  .pinMinor
  {
  cursor:pointer;text-decoration:none;
  z-index:5;
  }

  .pinModerate
  {
  cursor:pointer;text-decoration:none;
  z-index:5;
  }

  .pinMajor
  {
  cursor:pointer;text-decoration:none;
  z-index:5;
  }

</style>

<script type="text/javascript" language="javascript">
  var map = null;
  var myPanel;

  function OnPageLoad()
  {
    // Latitude and longitude have been hard-coded
    var params = new Object();
    params.latitude = 42.42;
    params.longitude = -83.02;
  params.zoomlevel = 10;
  params.mapstyle = Msn.VE.MapStyle.Road;
  params.showScaleBar = false;
  params.showDashboard = true;
  params.dashboardSize = Msn.VE.DashboardSize.Normal;
  params.dashboardX = 5;
  params.dashboardY = 5;
  map =
    new Msn.VE.MapControl(document.getElementById("myMap"),
     params);
  map.Init()
  PopulateMap();
}

function PopulateMap()
{
  // ZIP code has been hard-coded
  var zip = "48067";
  var mapmiles = 5;
  var mapseverity = 0;

  Anthem_InvokePageMethod(
    'PopulateMap', [zip, mapmiles, mapseverity], CallBack);
}

function CallBack(result)
{
  var table = result.value;
  var title = '';
  var description = '';
  var severity = '';

  var popstr = "";
  var newPushpin = null;
  var categry = '';

  for(var i=0;i<table.Rows.length;i++)
  {
    description = table.Rows[i].description ;
    description = replaceAll(description, ",", "<br>");
    severity = table.Rows[i].severity;
  title = table.Rows[i].title;
  category = table.Rows[i].category;
  severity = severity.replace(/^\s*|\s*$/g,"");
  category = category.replace(/^\s*|\s*$/g,"");
```

```
description = escape(description);
title = escape(title);

switch(category)
{
  case "Construction":
    var message = '<div><img src=\"pins/dude.gif\"';
      message +=  'onmouseover=\"popup(\'' + title + '\',\''
      message += description + '\','+  table.Rows[i].latitude
      message += ', ' + table.Rows[i].longitude
      message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
      message += '</div>';
    map.AddPushpin('pin' + i, table.Rows[i].latitude,
     table.Rows[i].longitude, 20, 20, 'pinConstruction', message);
    break;

    case "Incident":
      switch(severity)
      {
        case "Minor":
          var message = '<div><img src=\"pins/minor.gif\"';
            message +=  'onmouseover=\"popup(\'' + title + '\',\''
            message += description + '\','+  table.Rows[i].latitude
            message += ', ' + table.Rows[i].longitude
            message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
            message += '</div>';

        map.AddPushpin('pin' + i, table.Rows[i].latitude,
         table.Rows[i].longitude, 88, 34, 'pinMinor', message);
        break;

      case "Moderate":
        var message = '<div><img src=\"pins/moderate.gif\"';
          message += 'onmouseover=\"popup(\'' + title + '\',\''
          message += description + '\','+  table.Rows[i].latitude
          message += ', ' + table.Rows[i].longitude
          message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
          message += '</div>';

        map.AddPushpin('pin' + i, table.Rows[i].latitude,
         table.Rows[i].longitude, 88, 34, 'pinModerate', message);
        break;
```

```
            case "Major":
              var message = '<div><img src=\"pins/major.gif\"';
                message +=  'onmouseover=\"popup(\'' + title + '\',\''
                message += description + '\','+  table.Rows[i].latitude
                message += ', ' + table.Rows[i].longitude
                message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
                message += '</div>';

              map.AddPushpin('pin' + i, table.Rows[i].latitude,
               table.Rows[i].longitude, 88, 34, 'pinMajor', message);
              break;
          }
        }
      }
    }

    function replaceAll( str, from, to )
    {
      var newindex = str.indexOf( from );
      while ( newindex > -1 )
      {
        str = str.replace( from, to );
        newindex = str.indexOf( from );
      }
      return str;
    }
  </script>
</head>

<body onload="OnPageLoad();">
  <form id="form1" runat="server">
    <div id="myMap" style="WIDTH: 600px; HEIGHT: 400px;
      OVERFLOW:hidden;"></div>
  <div id="popup"></div>
  </form>


  <script type="text/javascript">
    offX = -20;
    offY = 15;
    var popupStyle;
    var varY = -999;
    // Browser check variables
    var ns4 = document.layers;
    var ns6 = document.getElementById&&!document.all;
    var ie4 = document.all;
    var iex = (document.all);
```

```
if (ns4)
{
  popupStyle = document.popup;
}
else if (ns6)
{
popupStyle = document.getElementById("popup").style;
}
else if (ie4)
{
popupStyle = document.all.popup.style;
}

if(ns4)
{
document.captureEvents(Event.MOUSEMOVE);
}
else
{
popupStyle.visibility = "visible";
popupStyle.display = "none";
}

function popup(title, text, latit, longit)
{

  text = unescape(text);
  title = unescape(title);
  title = "<strong>" + title + "</strong><br>";

  var popHTML = "<TABLE  WIDTH=200 BORDER=1 BORDERCOLOR=black
   CELLPADDING=1 CELLSPACING=0 BGCOLOR=white";
  popHTML += "><TR><TD ALIGN=center><FONT COLOR=black SIZE=2>" +
   title + text + "</FONT></TD></TR></TABLE>";

  var left = map.GetX(longit);
  var top = map.GetY(latit);
  popupStyle.left = left + offX;
  popupStyle.top = top + offY;

  if(ns4)
  {
    popupStyle.document.write(popHTML);
    popupStyle.document.close();
    popupStyle.visibility = "visible"
  }
```

```
        if(ns6)
        {
          document.getElementById("popup").innerHTML = popHTML;
          popupStyle.display = '';
        }
        if(ie4)
        {
          document.all("popup").innerHTML = popHTML;
          popupStyle.display = '';
        }
      }

      function closePopup()
      {

        varY = -999;

        if(ns4)
        {
          popupStyle.visibility = "hidden";
        }
        else if (ns6||ie4)
        {
          popupStyle.display = "none";
        }
      }
    </script>
  </body>
</html>
```

Default.aspx.cs

Here's the code-behind file:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
```

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Anthem.Manager.Register(this);
  }

  [Anthem.Method]
  public DataTable PopulateMap(string zipCode, int miles, int severity)
  {

    string urlpart1 = "http://maps.yahoo.com/traffic.rss?csz=";
    string urlpart2 = "&mag=";
    string urlpart3 = "&minsev=";
    DataTable returnDataTable = new DataTable();

    // Make the call and do map
    DataSet ds = new DataSet();
    string urlComplete = urlpart1 + zipCode + urlpart2 + miles.ToString()
     + urlpart3 + severity.ToString();

    ds.ReadXml(urlComplete);
    DataTable rsstbl = ds.Tables[2];

    returnDataTable.Columns.Add("latitude");
    returnDataTable.Columns.Add("longitude");
    returnDataTable.Columns.Add("title");
    returnDataTable.Columns.Add("description");
    returnDataTable.Columns.Add("severity");
    returnDataTable.Columns.Add("category");

    DataRow buildrow = null;
    foreach (DataRow row in rsstbl.Rows)
    {
      buildrow = returnDataTable.NewRow();
      buildrow["title"] = row["title"];
      buildrow["description"] = row["description"];
      buildrow["severity"] = row["severity"];
      buildrow["category"] = row["category"];

      string[] foundrow = row["link"].ToString().Split('&');
      foreach (string strPoints in foundrow)
      {
        if (strPoints.IndexOf("mlt=") > -1)
        {
          string[] strLat = strPoints.Split('=');
          buildrow["latitude"] = strLat[1];
        }
        if (strPoints.IndexOf("mln=") > -1)
        {
          string[] strLon = strPoints.Split('=');
          buildrow["longitude"] = strLon[1];
        }
      }
      returnDataTable.Rows.Add(buildrow);
    }
    return returnDataTable;
  }
}
```

There really isn't much to the page, so it won't take long to dissect the application. As I stated earlier in the chapter, this application is a mash-up of three technologies, Virtual Earth, Yahoo's traffic feed, and Ajax.

Microsoft's Virtual Earth

One of the first things that I feel I should mention about this particular tool is that it is free **ONLY** if you use it for noncommercial purposes. If you use it for any for-profit purpose, you will need to use the commercial install instead. In our current example, we'll be using the free version of the control. You can find the appropriate MSDN information for it at `http://msdn.microsoft.com/library/default.asp?url=/library/en-us/VEMCSDK/HTML/Introduction.asp`.

Once you've opted for the appropriate license, you'll notice that Virtual Earth is really just a few files that you'll reference from your client-side application. As you can see from the first few lines of the `.aspx` code, we establish the VE control references right from the start:

```
<link href="http://dev.virtualearth.net/standard/v2/MapControl.css"
 type="text/css" rel="stylesheet" />
<script src="http://dev.virtualearth.net/standard/v2/MapControl.js"
 type="text/javascript"></script>
```

After setting script locations, using the map is rather simple. To render a viable map, we must accomplish a few things:

- Instantiate and assign the maps parameters.
- Create an instance of the map, with those parameters.
- Add pushpins (if desired).

First and foremost, we must create a new instance of the map. We accomplish this as shown:

```
map = new Msn.VE.MapControl(document.getElementById("myMap"), params);
```

The constructor accepts the placement element (our `div` container) as well as a detailed list of settings, as parameters prior to rendering your requested area:

- *Latitude*
- *Longitude*

- *Zoom level*: `1` (country level) to `19` (street level)
- *Map style*: `r`—road, `h`—hybrid, and `a`—aerial. `Msn.VE.MapStyle` enumerations available, as well
- *showScaleBar*: A scaled comparison bar
- *showDashboard*: The control panel for the map
- *dashboardSize*: Normal or small
- *dashboardX*: Left location of the dashboard
- *dashboardY*: Top location of the dashboard

As most of those listed are self-explanatory, only map style needs a bit more of an explanation.

Map Styles

Each of the map styles provides a unique perspective, so you'll need to choose what is appropriate for your application.

Road View

Selecting `r` for this particular parameter will render a standard road map view, as shown in Figure 13-10.

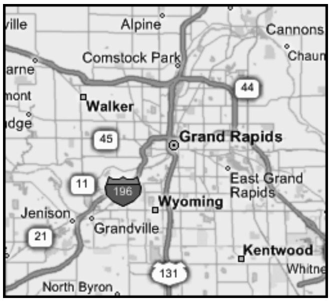


Figure 13-10. *Road view*

Hybrid View

Selecting h will produce the aerial view with labels, as you see in Figure 13-11.

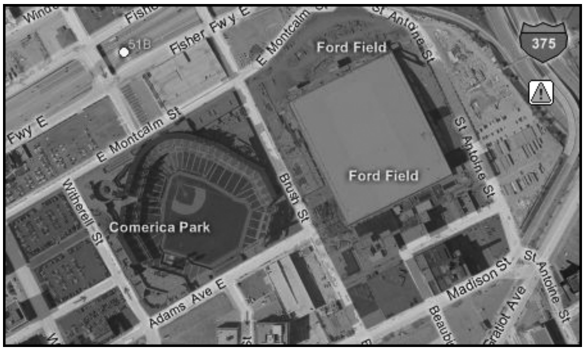


Figure 13-11. *Hybrid view*

Aerial View

An a setting for map style will render a map similar to the hybrid, but without the labels, as shown in Figure 13-12.

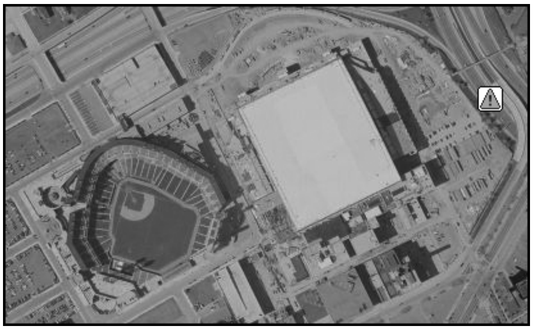


Figure 13-12. *Aerial view*

Pushpins

The only other mapping component that we have yet to discuss is the Virtual Earth pushpin. Figure 13-4 demonstrated the pushpin pop-up that we have access to, out of the box. Creating a pushpin requires only one method:

```
map.AddPushpin('pin' + i, table.Rows[i].latitude,
 table.Rows[i].longitude, 88, 34, 'pinMajor', message);
```

Here's the constructor for the pushpin:

```
AddPushpin(id, latitude, longitude, width, height, css class, innerHTML, Z-index)
```

- `id`: A unique ID for the pushpin
- `latitude`: Global location on the map to place the pushpin
- `longitude`: Global location on the map to place the pushpin
- `width`: Width of the pushpin
- `height`: Height of the pushpin
- `css class`: Class name of the CSS style to inherit
- `innerHTML`: HTML text for the mouseover pop-up
- `Z-index`: The Z-index of the pushpin

I've created a few CSS classes for each of the individual pushpins, but they all have the same contents. You could, of course, reduce them down to one and use that as the `class` parameter for the pushpin. I've spread them out for the sake of customization and readability. If you're not interested in pop-ups in your own Virtual Earth map, you'll make heavy use of the style blocks, perhaps assigning your own images to the `background-image` property. For now, the styles appear as shown:

```
.pinConstruction
{
cursor:pointer;text-decoration:none;
z-index:5;
}
.pinMinor
{
cursor:pointer;text-decoration:none;
z-index:5;
}
.pinModerate
{
cursor:pointer;text-decoration:none;
z-index:5;
}
.pinMajor
{
cursor:pointer;text-decoration:none;
z-index:5;
}
```

Now that we've examined the style sheets for pushpins, the actual creation and placement of those items simply involve retrieving the dynamic data (our traffic data), building the HTML for the pushpin, and adding them to the map, which you'll see in the "Using the Feed" section.

And this brings us to our second major technology of the mash-up, the Yahoo traffic feed.

Yahoo Traffic Feed

The kind folks at Yahoo have made available a rather intricate RSS feed, capable of accepting a variety of request parameters and returning any registered traffic reports for the given area. I should note, however, that not all cities are supported. The best thing you could do is just plug in some parameters and try it out. Also, Yahoo relies on the submission of reports from the appropriate Department of Transportation, and occasionally there may be things unreported. For instance, as I was driving home recently, I came across a construction zone, yet the Yahoo RSS feed showed nothing for my area. So if you're going to use the feed for retail purposes, be aware that it's not always spot-on. But it's a cool tool to use and worthy of deep scrutiny.

Fortunately for all you RSS addicts, the feed is RSS 2.0 conformant. That's good news for those of you building and implementing your own aggregators, as you'll be able to seamlessly integrate this feed.

Calling the feed can be as simple as pasting the URL into your browser and viewing the results. For instance, a feed of Detroit, Michigan, can be obtained by pasting the following URL into the address bar and letting Yahoo work its magic: `http://api.local.yahoo.com/MapsService/rss/trafficData.xml?appid=YahooDemo&city=Detroit&state=Michigan`.

Let's take a look at the feed that comes back from Yahoo, and then we will discuss the request parameters and response elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#" version="2.0">
<channel>
<title>Yahoo! Traffic Alerts</title>
<link>http://maps.yahoo.com</link>
<language>en</language>
<description>Traffic Alerts</description>
<copyright>Copyright (c) 2005 Yahoo! Inc. All rights reserved.</copyright>
<lastBuildDate>Tue, 21 Feb 2006 13:39:08 -0800</lastBuildDate>
<image>
<url>http://us.i1.yimg.com/us.yimg.com/i/us/nt/b/rss/rss_search.gif</url>
<title>Yahoo! Maps</title>
<link>http://maps.yahoo.com</link>
<width>142</width>
<height>18</height>
</image>
<item>
<title>Object on Roadway</title>
<link>http://img.maps.yahoo.com/mapimage?MAPDATA=
A3yNBed6wXWyUygjgNR64Ey8tpikQJ7Tr.w_72S_uZO2TXZQh
BIPCCtZdEUyHQp62zZ8za6IlWspuj1TooXO5Aqep9JPK6E3Z3
6Pc7V6v5KMSrM7DXxhaCBffkwmuA5oxi2_.vpu423r1eYCAAK
VEra.C.c2OGNu9ra9fYO-</link>
<description>DEBRIS ON ROAD; USE CAUTION; ON LEFT
 SIDE OF RAMP   STAY RIGHT   FIBERGLASS INSULATION
IS PARTIALLY BLOCKING THE RAMP.</description>
<geo:lat>42.487840</geo:lat>
<geo:long>-83.045690</geo:long>
```

```
<guid isPermaLink="false">http://img.maps.yahoo.com/mapimage?
MAPDATA=A3yNBed6wXWyUygjgNR64Ey8tpikQJ7Tr.w_72S_uZO2TXZQhBIPC
CtZdEUyHQp62zZ8za6IlWspuj1TooXO5Aqep9JPK6E3Z36Pc7V6v5KMSrM7DX
xhaCBffkwmuA5oxi2_.vpu423r1eYCAAKVEra.C.c2OGNu9ra9fYO-</guid>
<pubDate>Tue, 21 Feb 2006 13:05:41 -0800</pubDate>
</item>
</channel>
</rss>
<!-- ws02.search.re2.yahoo.com compressed/chunked Tue Feb 21 13:42:40 PST 2006 -->
```

What? Only one incident in the entire city of Detroit? OK, so maybe I deleted a few (or 15) for the sake of brevity. But the structure is the same, and you'll be able to understand the concept being presented here.

To get this feed, you'll notice that I supplied the RSS address with a few selective parameters:

```
appid=YahooDemo&city=Detroit&state=Michigan
```

The input parameters are key to getting the traffic for the region that you're interested in. Yahoo has been gracious enough to supply a pretty detailed list of the requisite items, provided here in Table 13-1.

Table 13-1. *Yahoo RSS Request Parameters*

Parameter	Data Type	Description
`appid`	String	Application ID of your site. Any unique ID that you specify.
`latitude`	Float (–90 to 90)	The latitude of the location.
`longitude`	Float	The longitude of the location.
`street`	String	The street name of the location—unit number optional.
`city`	String	The city name of the location.
`zip`	Int (int-int)	Five-digit ZIP code or five-digit plus four-digit extension. City and state will be ignored if they don't match the ZIP code specified.

`location`	Free text	Allows you to enter complete location information in various formats, as previously shown:
		`city, state`
		`city, state, zip`
		`zip`
		`street, city, state`
		`street, city, state, zip`
		`street, zip`
		If you supply any information in this field, it will take priority over any of the other individual location fields.

`severity`	Int (1 to 5)	The smallest severity level to return (1 is minor, 5 is severe).
`zoom`	Int (1 to 12)	Zoom level (similar to the map) indicating area to capture events for. A 1 would be street level, and 12 gathers country level. If you provide a `radius` parameter, Yahoo will ignore this parameter.

Parameter	Data Type	Description
`radius`	Float	The area in miles that Yahoo should provide traffic data for.
`include_map`	Binary	This parameter is always 1. Yahoo will always generate a map.
`image_type`	PNG or GIF	PNG is the default image format, but GIFs are available.
`image_height`	Int (10 to 2000)	Height of the image returned in pixels. Default is 500.
`image_width`	Int (10 to 2000)	Width of the image returned in pixels. Default is 620.

Once you're armed with the proper parameters, you can start to play around with the RSS service a little more.

A few notes about the various parameters:

- `appid` is anything that you want it to be. It should be unique across your multiple applications.
- If you use the `location` parameter, be aware that anything you provide in the other indicators (state, ZIP, street, etc.) will be overridden by the data contained within the `location` text. Usage is pretty simple: `http://api.local.yahoo.com/MapsService/rss/trafficData.xml?appid=YahooDemo&location=Hollywood,%20California`.

The XML elements of the response text have been annotated well by Yahoo, as you would expect.

Yahoo Response Elements

Tables 13-2 and 13-3 show the response elements contained in the RSS feed.

Table 13-2. *Top-level Element*

Element	Description
`rss`	Topmost element of the feed. Yahoo's traffic feed conforms to RSS 2.0 specs.

Example:

```
<rss xmlns:geo="http://www.w3.org/2003/01/geo/wgs84_pos#" version="2.0">
```

The `rss` element contains a `channel` element as its child element. The `channel` element contains child elements as described in Table 13-3.

Table 13-3. *Channel Elements*

Element	Description
`title`	Feed title.
`link`	The Yahoo map URL.
`language`	Language of the supplied traffic alert (English—`en`).
`description`	The feed description ("Traffic Alerts").
`copyright`	The Yahoo default copyright for the feed.
`lastBuildDate`	Date of last update for the report location.
`image`	The identification image for the feed. Typically, this is the Yahoo logo. We'll discuss the `image` element and its child elements in a moment.
`item`	An individual traffic report. We'll discuss the `item` element momentarily.

Here's an example:

```
<channel>
<title>Yahoo! Traffic Alerts</title>
<link>http://maps.yahoo.com</link>
<language>en</language>
<description>Traffic Alerts</description>
<copyright>Copyright (c) 2005 Yahoo! Inc. All rights reserved.</copyright>
<lastBuildDate>Tue, 21 Feb 2006 13:39:08 -0800</lastBuildDate>
<image>
...more about this, later.
</image>
<item>
...we'll discuss this in a moment.
</item>
</channel>
```

Image Elements

The `image` element describes the icon or graphic associated with the feed. Most likely it will be the Yahoo Search logo, because we are using the Yahoo Search engine to obtain the results. (I'm guessing on the association. I don't work at Yahoo, and they don't exactly say, but it sure seems obvious.) In the preceding XML result, if we copy and paste the image URL location, we should see the graphic shown in Figure 13-13.

Figure 13-13. *Image element*

The `image` element has the structure outlined in Table 13-4.

Table 13-4. *Image Elements*

Element	Description
`url`	Web address of the image
`title`	The title of the site that the feed is referring to
`link`	URL of the site that the feed is referring to
`width`	Width of the image in pixels
`height`	Height of the image in pixels

Here's an example:

```
<image>
<url>http://us.i1.yimg.com/us.yimg.com/i/us/nt/b/rss/rss_search.gif</url>
<title>Yahoo! Maps</title>
<link>http://maps.yahoo.com</link>
<width>142</width>
<height>18</height>
</image>
```

Item Elements

The `channel` element contains one final element block, which is encapsulated within `<item>` tags. `item` elements are the actual reports that we're after, and the tags are pretty much what would we have come to expect so far, as you can see in Table 13-5.

Table 13-5. *Item Elements*

Element	Description
`title`	The title of the alert.
`link`	The address of the available map for the alert. This will always be available.
`description`	The details of the traffic report.
`geo:lat`	The latitude of the report.
`geo:long`	The longitude of the report.
`guid`	The Yahoo-generated identifier for the report.
`pubDate`	Date and time of the traffic incident.

Here's an example:

```
<item>
<title>Object on Roadway</title>
<link>http://img.maps.yahoo.com/mapimage?MAPDATA=A3yNBed6wXWyUygjgNR64Ey8tpikQJ7Tr
.w_72S_uZO2TXZQhBIPCCtZdEUyHQp62zZ8za6IlWspuj1TooXO5Aqep9JPK6E3Z36Pc7V6v5KMSrM7DXxha
CBffkwmuA5oxi2_.vpu423r1eYCAAKVEra.C.c2OGNu9ra9fYO-</link>
<description>DEBRIS ON ROAD; USE CAUTION; ON LEFT SIDE OF RAMP   STAY RIGHT
FIBERGLASS INSULATION  IS PARTIALLY BLOCKING THE RAMP.</description>
```

```
<geo:lat>42.487840</geo:lat>
<geo:long>-83.045690</geo:long>
<guid isPermaLink="false">http://img.maps.yahoo.com/mapimage?MAPDATA=A3yNBed6wXWyU
ygjgNR64Ey8tpikQJ7Tr.w_72S_uZO2TXZQhBIPCCtZdEUyHQp62zZ8za6IlWspuj1TooXO5Aqep9JPK6E3Z
36Pc7V6v5KMSrM7DXxhaCBffkwmuA5oxi2_.vpu423r1eYCAAKVEra.C.c2OGNu9ra9fYO-</guid>
<pubDate>Tue, 21 Feb 2006 13:05:41 -0800</pubDate>
</item>
```

Using the Feed

So now that you have a pretty good grasp on what it is that Yahoo can do for you, let's take a look at how we bring this feed into the application and ultimately parse the data for the map. Here we will discuss the method from which we will draw our pushpin locations that will represent various traffic reports across the area. We handle the Ajax-enabled call to retrieve a data table in PopulateMap:

```
function PopulateMap()
{
  // ZIP code has been hard-coded
  var zip = "48067";
  var mapmiles = 5;
  var mapseverity = 0;

  Anthem_InvokePageMethod('PopulateMap',[zip, mapmiles, mapseverity], CallBack);

}
```

As you can see, we're hard-coding the ZIP code for the map that we wish to retrieve. We could just as easily provide a textbox interface for this element, but for simplicity's sake, we'll just hard-code it. We then provide a radius of coverage that we'd like to see returned as well as the severity level of the reports being generated. We'll ask for zero, which means we'll receive all the results for the area we've requested.

We then make the call to the Ajax method, providing CallBack as the callback function.

Server Side

On the server side of things, we communicate with Yahoo, providing the requisite parameters and building a DataTable that we'll pass back to the client for parsing:

```
[Anthem.Method]
public DataTable PopulateMap(string zipCode, int miles, int severity)
{
  string urlpart1 = "http://maps.yahoo.com/traffic.rss?csz=";
  string urlpart2 = "&mag=";
  string urlpart3 = "&minsev=";
  DataTable returnDataTable = new DataTable();
```

```
    // Make the call and do map
    DataSet ds = new DataSet();
    string urlComplete = urlpart1 + zipCode + urlpart2 + miles.ToString()
      + urlpart3 + severity.ToString();

    ds.ReadXml(urlComplete);
    DataTable rsstbl = ds.Tables[2];

    returnDataTable.Columns.Add("latitude");
    returnDataTable.Columns.Add("longitude");
    returnDataTable.Columns.Add("title");
    returnDataTable.Columns.Add("description");
    returnDataTable.Columns.Add("severity");
    returnDataTable.Columns.Add("category");

    DataRow buildrow = null;
    foreach (DataRow row in rsstbl.Rows)
    {
      buildrow = returnDataTable.NewRow();
      buildrow["title"] = row["title"];
      buildrow["description"] = row["description"];
      buildrow["severity"] = row["severity"];
      buildrow["category"] = row["category"];

      string[] foundrow = row["link"].ToString().Split('&');
      foreach (string strPoints in foundrow)
      {
        if (strPoints.IndexOf("mlt=") > -1)
        {
          string[] strLat = strPoints.Split('=');
          buildrow["latitude"] = strLat[1];
        }

        if (strPoints.IndexOf("mln=") > -1)
        {
          string[] strLon = strPoints.Split('=');
          buildrow["longitude"] = strLon[1];
        }
      }
      returnDataTable.Rows.Add(buildrow);
    }
    return returnDataTable;
  }
```

Client Side

Back on the client side of things, as you've seen before, our callback method waits patiently for the traffic table to return for parsing. After parsing the table and adding pushpins with the `map.AddPushpin()` method, we exit the process and wait for the next user interaction:

```
function CallBack(result)
{
  var table = result.value;
  var title = '';
  var description = '';
  var severity = '';

  var popstr = "";
  var newPushpin = null;
  var categry = '';

  for(var i=0;i<table.Rows.length;i++)
  {
    description = table.Rows[i].description ;
    description = replaceAll(description, ",", "<br>");
    severity = table.Rows[i].severity;
    title = table.Rows[i].title;
    category = table.Rows[i].category;
    severity = severity.replace(/^\s*|\s*$/g,"");
    category = category.replace(/^\s*|\s*$/g,"");

    description = escape(description);
    title = escape(title);
    switch(category)
    {
      case "Construction":
        var message = '<div><img src=\"pins/dude.gif\"';
          message +=  'onmouseover=\"popup(\'' + title + '\',\'';
          message += description + '\','+  table.Rows[i].latitude ;
          message += ', ' + table.Rows[i].longitude ;
          message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
          message += '</div>';
        map.AddPushpin('pin' + i, table.Rows[i].latitude,
          table.Rows[i].longitude, 20, 20, 'pinConstruction', message);
        break;

      case "Incident":
        switch(severity)
        {
          case "Minor":
            var message = '<div><img src=\"pins/minor.gif\"';
```

```
              message +=  'onmouseover=\"popup(\'' + title + '\',\''
              message += description + '\','+  table.Rows[i].latitude
              message += ', ' + table.Rows[i].longitude
              message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
              message += '</div>';

            map.AddPushpin('pin' + i, table.Rows[i].latitude,
              table.Rows[i].longitude, 88, 34, 'pinMinor', message);
            break;

          case "Moderate":
            var message = '<div><img src=\"pins/moderate.gif\"';
              message +=  'onmouseover=\"popup(\'' + title + '\',\''
              message += description + '\','+  table.Rows[i].latitude
              message += ', ' + table.Rows[i].longitude
              message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
              message += '</div>';

            map.AddPushpin('pin' + i, table.Rows[i].latitude,
              table.Rows[i].longitude, 88, 34, 'pinModerate', message);
            break;

          case "Major":
            var message = '<div><img src=\"pins/major.gif\"';
             message +=  'onmouseover=\"popup(\'' + title + '\',\''
             message += description + '\','+  table.Rows[i].latitude
             message += ', ' + table.Rows[i].longitude
             message += ')\"; ONMOUSEOUT=\"closePopup()\"; />';
             message += '</div>';

            map.AddPushpin('pin' + i, table.Rows[i].latitude,
              table.Rows[i].longitude, 88, 34, 'pinMajor', message);
            break;
        }
      }
    }
  }
}
```

Depending on the severity of the traffic incident, a corresponding set of parameters is generated for the AddPushpin() method. We supply the latitude and longitude as you would expect, and we also provide the generated message. Our message is built upon the idea that we'll be using a custom div and a nested pin image as the pushpin, rather than just a static image. Because of this, we'll be able to generate a pop-up of the incident details as shown earlier in Figure 13-4. The popup method is called in the onmouseover event, as shown previously. The popup code itself is a pretty easy read:

```
<div id="popup"></div>
<script type="text/javascript">
  offX = -20;
  offY = 15;
  var popupStyle;
  var varY = -999;
  // Browser check variables
  var ns4=document.layers;
  var ns6=document.getElementById&&!document.all;
  var ie4=document.all;
  var iex = (document.all);

  if (ns4)
  {
    popupStyle = document.popup;
  }
  else if (ns6)
  {
    popupStyle = document.getElementById("popup").style;
  }
  else if (ie4)
  {
    popupStyle = document.all.popup.style;
  }

  if(ns4)
  {
    document.captureEvents(Event.MOUSEMOVE);
  }
  else
  {
    popupStyle.visibility = "visible";
    popupStyle.display = "none";
  }

  function popup(title, text, latit, longit)
  {
    text = unescape(text);
    title = unescape(title);
    title = "<strong>" + title + "</strong><br>";

    var popHTML = "<TABLE  WIDTH=200 BORDER=1 BORDERCOLOR=black
      CELLPADDING=1 CELLSPACING=0 BGCOLOR=white";
    popHTML += "><TR><TD ALIGN=center><FONT COLOR=black SIZE=2>" +
      title + text + "</FONT></TD></TR></TABLE>";
    var left = map.GetX(longit);
    var top = map.GetY(latit);
    popupStyle.left = left + offX;
    popupStyle.top = top + offY;

    if(ns4)
    {
      popupStyle.document.write(popHTML);
      popupStyle.document.close();
      popupStyle.visibility = "visible"
    }

    if(ns6)
    {
      document.getElementById("popup").innerHTML = popHTML;
      popupStyle.display = '';
    }

    if(ie4)
    {
      document.all("popup").innerHTML = popHTML;
      popupStyle.display = '';
    }
  }

  function closePopup()
  {

    varY = -999;

    if(ns4)
    {
      popupStyle.visibility = "hidden";
    }
    else if (ns6||ie4)
    {
      popupStyle.display = "none";
    }
  }
</script>
```

We're able to supply the popup function parameters, based on information that we've retrieved from the Yahoo feed:

```
function popup(title, text, latit, longit)
```

The `popup` function builds and compiles the `innerHTML` that we'll push out to the empty `<div>` block that we have statically assigned to the body of the page:

```
<div id="popup"></div>
```

Our `popup` function basically turns on the visibility of the floating `<div>`, and our traffic details are proudly displayed to the user.

And that's it! We've successfully completed the full cycle of the map application.

Summary

Combining Ajax with other various online tools, whether they be dynamic mapping, RSS feeds, or image providers, can be truly powerful, with a huge impact on site appreciation. Ajax-enabled applications, as you've seen here, can be so much more than just simple dynamic textbox updating. It's easy to get excited when you see how easy it is to mash up your own applications.

In the next few chapters, you'll see how Ajax can be used, not as a mash-up component, but as an integral part of application frameworks as well. We'll build an Ajax Web Part for ASP.NET 2.0 as we begin to take advantage of the unique, new tools that version 2.0 has given us.