The Bing Maps AJAX Control 7.0 is essentially a JavaScript control that contains everything you need to display your maps and overlays with Bing Maps using a browser. If you have ever done any work with Google Maps API, it is very similar and you will right at home using this control! This post will walk you through creating a simple map search and site results screen using the Bing Maps control and using SQL Server 2008+. Below is a picture of the end result:

Bing Maps AJAX Control 7.0 Example

Note, the complete reference material for the Bing Maps AJAX Control can be found here (<http://msdn.microsoft.com/en-us/library/gg427610>) and this demonstration was formed using this material as a basis.

Now the first thing, and most important, is that you need to create an account and generate a Bing Maps API Key. To do so, follow this link ( <http://www.bingmapsportal.com/>). Make sure you generate a ‘Developer’ key for your testing and R&D purposes. Hold onto this key, you will need it later.

First, we need to set up the database.  You will want to store all the locations in a database table.  So let’s create this table in SQL Server 2008/R2/2012 by using code like the below:

Note, this example is making use of SQL Server’s spatial data types. Please read this article if you would like some background info on these (<http://stefanzvonar.com/2012/08/07/spatial-data-types-in-sql-server-and-sql-example/>)

[sourcecode language="sql"]  
CREATE TABLE [dbo].[tblLocation](

[LocationID] [int] IDENTITY(1,1) NOT NULL,

[LocationDescription] [nvarchar](100) NOT NULL,

[Address1] [nvarchar](50) NOT NULL,

[Address2] [nvarchar](50) NULL,

[Town] [nvarchar](50) NOT NULL,

[County] [nvarchar](50) NULL,

[Postcode] [nvarchar](10) NOT NULL,

[Country] [nvarchar](50) NOT NULL,

[Latitude] [decimal](10, 7) NOT NULL,

[Longitude] [decimal](10, 7) NOT NULL,

[SRID] [int] NOT NULL,

-- Note: In SQL Server 2012, you can have this computed columns based on spatial types as PERSISTED, for faster retrieval and calculations.

[Point] AS ([geography]::Point([Latitude],[Longitude],[SRID])),

CONSTRAINT [PK\_tblLocation] PRIMARY KEY CLUSTERED

(

[LocationID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, FILLFACTOR = 80) ON [PRIMARY]

) ON [PRIMARY]

GO

ALTER TABLE [dbo].[tblLocation] ADD CONSTRAINT [DF\_tblLocation\_Latitude] DEFAULT ((0)) FOR [Latitude]

GO

ALTER TABLE [dbo].[tblLocation] ADD CONSTRAINT [DF\_tblLocation\_Longitude] DEFAULT ((0)) FOR [Longitude]

GO

[/sourcecode]

You can then pre-populate this table with your store locations. Here is an SQL example:

[sourcecode language="sql"]  
insert into dbo.tblLocation(LocationDescription, Address1, Address2, Town, County, Postcode, Country,

Latitude, Longitude, SRID)

values('CBD Mall', 'Queen Street Mall', 'Queen Street', 'Brisbane', 'Queensland', '4000', 'Australia',

-27.4709331, 153.02350239999998, 4326)

insert into dbo.tblLocation(LocationDescription, Address1, Address2, Town, County, Postcode, Country,

Latitude, Longitude, SRID)

values('Valley Mall', 'Brunswick Street Mall', 'Brunswick Street', 'Fortitude Valley', 'Queensland', '4006', 'Australia',

-27.4568523, 153.03294370000003, 4326)

insert into dbo.tblLocation(LocationDescription, Address1, Address2, Town, County, Postcode, Country,

Latitude, Longitude, SRID)

values('Brisbane Airport', 'Brisbane Domestic Terminal', 'Brisbane Airport', 'Brisbane', 'Queensland', '4009', 'United Kingdom',

-27.3849801, 153.12056069999994, 4326)

[/sourcecode]

Note, if you do not know the actual latitudes and longitudes for your locations, you can use a bulk geocoding service (if you have a lot of addresses to find coordinates for). An example of this is Bing’s Geocode Dataflow API (<http://msdn.microsoft.com/en-us/library/ff701733.aspx>), part of Microsoft’s Bing Spatial Data Services, which allows you to use XML or CSV files for bulk geocoding. Another bulk geocoding service that I found is this one (<https://www.geocoderpro.com/en/resources/batch-geocoding/>), although I have not formally tested it out yet.

However, if you only have a small amount of addresses to geocode, you can also simply use the standard Bing or Google Maps API as well. The following link describes how to geocode a single address into coordinates using the Bing Maps API (<http://stefanzvonar.com/2012/07/17/obtain-latitude-and-longitude-co-ordinates-for-an-address-using-the-bing-maps-ajax-control-7-0-and-asp-net-controls/>).

Now that you have a locations table and some location data, you will want to create a stored procedure to find all the locations based on some parameters. This will use SQL Server spatial data type methods to find the nearest locations based on a passed in co-ordinate. Here is the basic procedure code:

[sourcecode language="sql"]  
CREATE PROCEDURE [dbo].[proc\_Location\_List\_By\_Geography]

(

@dmlLat decimal(10, 7),

@dmlLng decimal(10, 7),

@intRadiusKm int

)

As

SET NOCOUNT ON

-- Assuming a WGS84 projection, but change 4326 to the appropriate spatial reference ID for your stored coordinates

DECLARE @SearchPoint as geography

SET @SearchPoint = geography::Point(@dmlLat, @dmlLng, 4326)

-- The STDistance instance method returns a measurement based on the [unit\_of\_measure] in [sys].[spatial\_reference\_systems] table for the appropriate spatial reference.

-- In most cases, this is in metres.

SELECT LocationID, LocationDescription, Address1, Address2, Town, County, Postcode, Latitude, Longitude, SRID, (Point.STDistance(@SearchPoint)/1000) AS DistanceKm

FROM tblLocation

WHERE Point.STDistance(@SearchPoint) <= (@intRadiusKm \* 1000)

-- Return XML for formatting results

FOR XML RAW('marker'),ROOT('markers')

SET NOCOUNT OFF

GO

[/sourcecode]

Executing the above stored procedure will render XML like the below:

[sourcecode language="xml"]  
<markers>

<marker LocationID="1" LocationDescription="CBD Mall" Address1="Queen Street Mall" Address2="Queen Street" Town="Brisbane" County="Queensland" Postcode="4000" Latitude="-27.4709331" Longitude="153.0235024" SRID="4326" DistanceKm="0.000000000000000e+000" />

<marker LocationID="2" LocationDescription="Valley Mall" Address1="Brunswick Street Mall" Address2="Brunswick Street" Town="Fortitude Valley" County="Queensland" Postcode="4006" Latitude="-27.4568523" Longitude="153.0329437" SRID="4326" DistanceKm="1.818090141984348e+000" />

<marker LocationID="3" LocationDescription="Brisbane Airport" Address1="Brisbane Domestic Terminal" Address2="Brisbane Airport" Town="Brisbane" County="Queensland" Postcode="4009" Latitude="-27.3849801" Longitude="153.1205607" SRID="4326" DistanceKm="1.352086214866322e+001" />

</markers>  
[/sourcecode]

That completes the SQL Server side of it. Now it is time to create some basic ASP.NET pages to show the location results.

For the web side of things, you will essentially want two pages 1) A Search Request Page and 2) A Search Response Page.

Here is a snippet of code that demonstrates how the Search Request Page could look like (in the source code provided, this is MapExample.aspx):

[sourcecode language="html"]  
 <div style="margin-left:20px;">

<div style="margin-left: 5px;">

<span style="font-size: 10px;">Please enter an address, town or post code and click search</span>

<br />

Address: <input type="text" id="txtSearchAddress" value="Edward Street, Brisbane" />&nbsp;

Country: <input type="text" id="txtSearchCountry" value="Australia" />&nbsp;

Distance:

<select id="ddlRadius">

<option value="5">5 km</option>

<option value="10">10 km</option>

<option value="25">25 km</option>

<option value="50" selected>50 km</option>

<option value="100">100 km</option>

</select>&nbsp;

<input type="button" onclick="searchLocations()" value="Search"/>

</div>

<br />

<div style="clear:both; margin-left: 5px;">

<div id="sidebar" style="overflow: auto; height: 400px; width:150px; font-size: 11px; color: #000; float:left; margin-left:5px; padding-left:5px;">Search Results:</div>

<div style="float:left; margin-left: 5px;">

<div id="BingMap\_Div\_Container">

<div id="map" style="position:relative;width:750px;height:400px;"></div>

</div>

</div>

</div>

<br style="clear:both;" />

</div>

[/sourcecode]

On this page, you will need to reference some javascript to be able to connect to the Bing Maps service and process the results. In the solution code provided, this is bundled up in to the MapExample.js and Util.js files.

The Util.js file is mainly a bunch of handy functions to help with calling a rest service and processing a response dynamically. I actually found most of these functions from code samples from the Google Maps API (<http://code.google.com/p/gmaps-samples-v3/source/browse/trunk/xmlparsing/util.js?r=14>). In particular, for the ease of demonstration purposes, our search results come back from SQL Server as XML, and so these functions parse an XML response. You could go all fancy and make a SQLCLR procedure that returns JSON if you like and handle JSON instead. Perhaps one day SQL Server will be able to inherently return JSON result sets (hint hint SQL team).

Okay, so back to the Bing Maps API – below is the crux of the Javascript used to interact with Bing Maps. Of particular note is changing the credentials variable to use your own Bing API Key (generated at the start of this article). If you have done any online mapping tasks before, a lot of this code will be very familiar. One thing that is important to note, is that you have to add your pins to the map.entities collection (if you want to reference them later for event handling). This code example does just that, it adds the pins and info boxes to the entities collection. It is also important to note that there is just one collection for all your entity objects, so you need to know which ones are where (in this example, pins alternate with infoboxes, so I needed to ensure I invoke the click method of every second entity object when clicking on an item in the left menu item). Review the code and run the example for all that to make more sense :)

[sourcecode language="javascript"]  
// Bing Map API Javascript, tightly coupled with MapExample.aspx

var credentials = "YabbaDabbaDoo"; // Change this to your Bing API Key

var map;

var mapOptions;

var sbar;

var startingLat = -25; // view of Australia

var startingLng = 136; // view of Australia

var startingZoom = 4;

// Function to intialise map and side bar

function GetMap() {

mapOptions = {

credentials: credentials,

mapTypeId: Microsoft.Maps.MapTypeId.road, // try using Microsoft.Maps.MapTypeId.aerial for some cool imagery

zoom: startingZoom,

center: new Microsoft.Maps.Location(startingLat, startingLng)

};

map = new Microsoft.Maps.Map(document.getElementById('map'), mapOptions);

sbar = document.getElementById('sidebar');

}

// Function used when search button is clicked

function searchLocations() {

var address = document.getElementById('txtSearchAddress').value;

var country = document.getElementById('txtSearchCountry').value;

var searchString = address + ', ' + country;

// Make REST call to get geocoded information (that is, the coordinates for the address)

// This will use your call back procedure and return result in JSON format

var geocodeRequest = "http://dev.virtualearth.net/REST/v1/Locations?query=" + encodeURI(searchString) + "&output=json&jsonp=searchLocationsCallBack&key=" + credentials;

CallRestService(geocodeRequest);

}

// Function that clears the map with a message in the sidebar

function clearMap(msg) {

sbar.innerHTML = msg;

map = new Microsoft.Maps.Map(document.getElementById('map'), mapOptions);

}

// Callback function from geocoding request

function searchLocationsCallBack(result) {

// Check result was returned by gecode request

if (result &&

result.resourceSets &&

result.resourceSets.length > 0 &&

result.resourceSets[0].resources &&

result.resourceSets[0].resources.length > 0) {

searchLocationsNear(result);

}

else {

alert('Address not found');

clearMap('No results found. Please try a different address.');

}

}

// Grunt work function that finds the nearby locations

function searchLocationsNear(result) {

var lat = result.resourceSets[0].resources[0].point.coordinates[0];

var lng = result.resourceSets[0].resources[0].point.coordinates[1];

var radius = document.getElementById('ddlRadius').value;

var searchUrl = 'SearchResponse.aspx?lat=' + lat + '&lng=' + lng + '&radius=' + radius;

downloadUrl(searchUrl, function (data) {

var locationDB;

var locationAry = [];

var startRectangle;

var viewBoundaries;

//clear side bar entries

sbar.innerHTML = '';

if (data.documentElement == null) {

clearMap('No results found. Please try widening your search area.');

return;

}

// fetch locations from xml returned by database

locationDB = data.documentElement.getElementsByTagName('marker');

if (locationDB.length == 0) {

clearMap('No results found. Please try widening your search area.');

return;

}

// loop through found locations in the database

for (var i = 0; i < locationDB.length; i++) {

var address1 = locationDB[i].getAttribute('Address1');

var address2 = locationDB[i].getAttribute('Address2');

var town = locationDB[i].getAttribute('Town');

var postcode = locationDB[i].getAttribute('Postcode');

var distance = parseFloat(locationDB[i].getAttribute('DistanceKm'));

var location = new Microsoft.Maps.Location(parseFloat(locationDB[i].getAttribute('Latitude')), parseFloat(locationDB[i].getAttribute('Longitude')));

// push location to array for later use for map viewing

locationAry.push(location);

// Create pins on the map and the side bar entries

createPushPin(location, address1, address2, town, postcode);

createSidebarEntry(i, address1, address2, town, distance);

}

// Let microsoft figure out the viewing boundary to contain all the found locations

viewBoundaries = Microsoft.Maps.LocationRect.fromLocations(locationAry);

map.setView({ bounds: viewBoundaries });

});

}

// Create the pushpin with address information

function createPushPin(location, address1, address2, town, postcode) {

var content;

if (address2 == null) {

content = address1 + '<br/>' + town + '<br/>' + postcode;

}

else {

content = address1 + ', ' + address2 + '<br/>' + town + '<br/>' + postcode;

}

var pin = new Microsoft.Maps.Pushpin(location);

var pinInfoBox = new Microsoft.Maps.Infobox(location, { title: address1, visible: false, showPointer: true, offset: new Microsoft.Maps.Point(0, 20), description: content });

// Add event handler so that clicking the pin displays information about the location

Microsoft.Maps.Events.addHandler(pin, "click", function () {

pinInfoBox.setOptions({ visible: true });

});

// Add pins and information boxes to the map entities

// It is important to note that both pins and infoboxes are stored in the same collection. Thus, pins and infoboxes alternate in the collection.

map.entities.push(pin);

map.entities.push(pinInfoBox);

}

// Create the side bar entry as a menu item

function createSidebarEntry(index, address1, address2, town, distance) {

var div = document.createElement('div');

var address;

if (address2 == '' || address2 == null) {

address = address1 + '<br/>' + town;

}

else {

address = address1 + '<br/>' + address2 + '<br/>' + town;

}

var html = '<b>' + distance.toFixed(2) + ' kms: </b><br/>' + address;

div.innerHTML = html;

div.style.cursor = 'pointer';

div.style.marginBottom = '5px';

// Add event listeners to sidebar entries

// Note, pins are every second item in the entities collection, so multiple index by 2 for each pin

div.addEventListener("click", function () { var i = index \* 2; var pin = map.entities.get(i); Microsoft.Maps.Events.invoke(pin, 'click', ''); }, false);

// Pretty colour events

div.addEventListener("mouseover", function () { div.style.backgroundColor = '#eee'; }, false);

div.addEventListener("mouseout", function () { div.style.backgroundColor = '#fff'; }, false);

sbar.appendChild(div);

}

[/sourcecode]

When the page loads for the first time, we want to make sure we load up the map JavaScript variables. So in the code-behind of this SearchRequest.aspx page, we will add something like the below in the page load event handler:

[sourcecode language="vb"]  
 Protected Sub Page\_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

Page.ClientScript.RegisterStartupScript(Me.GetType(), "GetMap", "<script language='javascript'> GetMap(); </script>")

End Sub

[/sourcecode]

Okay, so all that is left is the guts for the page that will return the XML. If you create a new page (e.g. SearchResponse.aspx) in the same directory as the Search Request page, then all it needs to do is call the stored procedure that will return the locations as XML and return an XML response. The aspx will be empty except for the page declaration. The code-behind of this Search Response Page will look like the below:

[sourcecode language="vb"]  
Imports System.Xml

Imports System.Data.SqlClient

Partial Class SearchResponse

Inherits System.Web.UI.Page

' Page Load

Protected Sub Page\_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

Dim dmlSearchLatitude As Double

Dim dmlSearchLongitude As Double

Dim intSearchRadius As Int32

If Request.QueryString("lat") <> "" Then

dmlSearchLatitude = CDbl(Request.QueryString("lat"))

End If

If Request.QueryString("lng") <> "" Then

dmlSearchLongitude = CDbl(Request.QueryString("lng"))

End If

If Request.QueryString("radius") <> "" Then

intSearchRadius = CInt(Request.QueryString("radius"))

End If

' Here you make the call to your locations stored procedure (or data layer method, or however you have decided to architect your system)

Dim connDB = New SqlConnection

Dim cmd As New SqlCommand

connDB.ConnectionString = "Server=(local);Database=MapExample;Trusted\_Connection=True;" ' Change your database connection string to your own!

connDB.Open()

cmd.Connection = connDB

cmd.CommandType = Data.CommandType.StoredProcedure

cmd.CommandTimeout = 20

cmd.CommandText = "dbo.proc\_Location\_List\_By\_Geography"

cmd.Parameters.AddWithValue("@dmlLat", dmlSearchLatitude)

cmd.Parameters.AddWithValue("@dmlLng", dmlSearchLongitude)

cmd.Parameters.AddWithValue("@intRadiusKm", intSearchRadius)

' Execute the stored procedure and return the result as plain XML

Dim rdrXMLLocations As XmlReader = Nothing

rdrXMLLocations = cmd.ExecuteXmlReader()

Response.Expires = 0

Response.ContentType = "text/xml"

Dim oDocument As New XmlDocument()

Dim sb As New System.Text.StringBuilder()

Using rdrXMLLocations

While Not rdrXMLLocations.EOF

rdrXMLLocations.MoveToContent()

sb.Append(rdrXMLLocations.ReadOuterXml())

End While

rdrXMLLocations.Close()

End Using

If sb.ToString() <> "" Then

oDocument.LoadXml(sb.ToString())

End If

oDocument.Save(Response.Output)

Response.OutputStream.Flush()

Response.OutputStream.Close()

End Sub

End Class

[/sourcecode]

Hopefully that should be enough to get the Bing Maps AJAX Control 7.0 in action for you.

Click [here](https://www.box.com/s/336880a0e47819168eaa) to download a zip file of the example code used in this article.

<https://www.box.com/s/lkvfqy89ud7gioct3si5>

Hope this helps,

Stefan

-  
stefanzvonar.com – A collection of articles on computer programming, software development, data analysis and other technical tidbits.  
-