Bing Maps V7 – Modular design and Client Side Clustering

Last fall Microsoft released V7 of the Bing Maps AJAX control. This control is a complete redesign of the previous versions. This was done to make the map control smaller and faster. How much smaller and faster you ask? V7 is 1/3rd the size of V6.3 and renders up to 3x faster. As a result we can load much more data into the map than we ever could before.

**So how did they do it?**

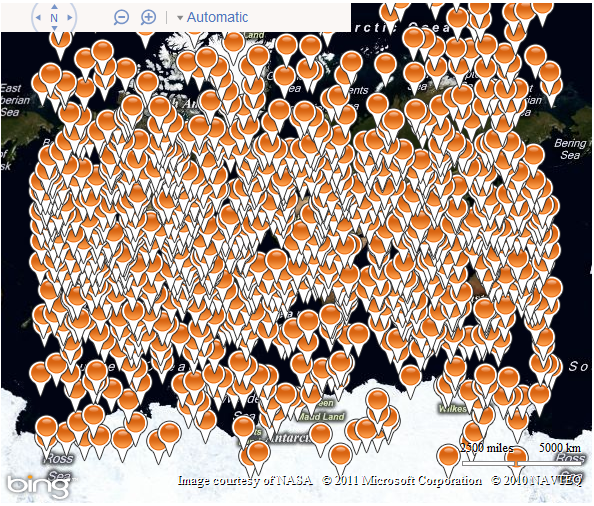
The new architecture of V7 is a modular design which means that we can build functionalities that plug into the map control to add functionality without having to include that functionality in all of our applications. This is a much better approach than what was done in V6.3 as this means that we can create applications that only load in the functionalities they need and not all the functionalities that are available. A good example of this is the routing functionality in Bing Maps. This is a key functionality in a lot of locator based applications however if you were to create an application that displayed business intelligence information on the map you likely don’t need this functionality, so why should you have to download it.

**So what does this mean going forward?**

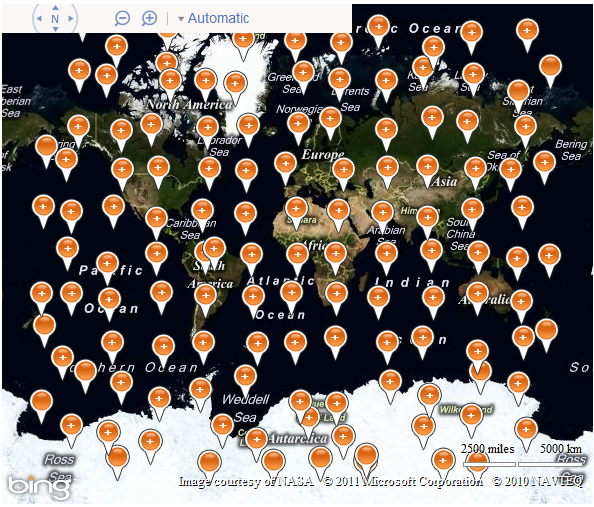
It means that we can create blocks of reusable code of key functionalities that can be easily plugged into our applications. Once a modular plugin is created it can be reused again in the future to speed up develop. This is very similar to jQuery plugins. This article will give an overview of what client side clustering is and show how to implement the client side clustering modular plugin for Bing Maps v7 that I’ve made available [here](http://cid-0d35222484a76a01.office.live.com/embedicon.aspx/BingMapsv7/BMv7.ClientSideClustering.zip).

**What is Client Side Clustering?**

Often we want to load a lot of data on the map, for example if you were a real estate company you would likely want to load up all the properties that matched a user’s search. This could potentially be a couple hundred, if not thousands of results. If you are zoomed in close to the map it will be pretty easy to identify each location on the map, but if you zoom out the pushpins will likely overlap and the map will become cluttered. Here is an example of 1000 pushpins that are unclustered:



Clustering is the process of grouping closely positioned locations together and representing them with a single pushpin. When you zoom in those locations will become further apart and will separate into their own pushpins. The client side part of “Client Side Clustering” is clustering the data on the fly in JavaScript rather than going back to the server to request more data. When you have a few thousand or less location to display on the map client side clustering can be significantly faster than server side clustering and also cuts down on request to your server thus making your application more scalable. Here is an example of the same 1000 pushpins from before being clustered:



**Background on Client Side Clustering and Bing Maps**

In the fall of 2008 I had put together an MSDN article on [Client Side Clustering](http://msdn.microsoft.com/en-us/library/cc161072.aspx) in V6. This article became very popular but required a lot of coding to implement. As a result of client demand Microsoft made this a built-in feature in V6.3. With the redesigned V7 Bing Maps control client side clustering is no longer a built in functionality. The client side clustering algorithm used in this article and provided in the sample code is an enhanced version of the algorithm in the MSDN article. This new algorithm has been optimized for performance and designed for easy reuse. It is recommend for use with data sets of 5000 locations or less.

**Grid Based Clustering**

Both algorithms use what is called grid based clustering. Grid based clustering consists of breaking the map view into a grid with evenly sized cells. Locations are then grouped into the grid cells in which they fall in. If there are more than one location in a grid cell then this is a cluster and is grouped together, otherwise a single location is displayed. Using Bing Maps the following approach is used to cluster data;

1. Determine the size of the map.
2. Use the size of the map and the gird size to determine the number of cells needed to fill the map view.
3. Create an empty array that has the same number of grids cells as calculated in step 2.
4. Loop through the data and do the following:
   1. Calculate the pixel coordinate of each location.
   2. Determine if pixel is within the bounds of the map.
   3. If location is within the bounds of the map, calculate the grid cell key that it falls in and add it to that cell in the array created in step 2.
5. Loop through all cells in the array created in step 2 and do the following:
   1. Determine if there is only one location in a cell and if there is then create a pushpin for that location.
   2. If there is more than one location for a cell then calculate a location to position the cluster and generate a clustered pushpin.
   3. Add pushpins to the map.

**Representing a Cluster**

When representing cluster with a pushpin a coordinate needs to be used. In the code I have provided there are three different ways to do this. The first method is to calculate the mean average coordinate of all the locations in the cluster. This gives a weighted position that most accurately represents the cluster; however it requires the most processing out of the three methods. The second method consists of representing the cluster by adding a pushpin to the center of the grid cell. This is less significant than the mean average positioning method but is a common method to use as it requires less processing. The third method is to use the location of the first location in the cluster. This ensures that the clustered pin falls on at least one location in the cluster and requires the least amount of processing power.

**Implementing the Modular plugin**

The client side clustering modular plugin consists of a class called *ClusteredEntityCollection* that wraps the EntityCollection class. This class will only work with point based data. One requirement is that all data objects that are passed into this class must have a Latitude and Longitude property. When initializing a new instance of the *ClusteredEntityCollection* class a reference to a map and an object containing clustering options are passed in.

|  |
| --- |
| var x = new ClusteredEntityCollection(map, options); |

During initialization an instance of an EntityCollection is created and added to the map and a *viewchangend* event is attached to a clustering function. Below is an outline of the modular *ClusteredEntityCollection* class. Additional methods and logic are added to this class to make this a complete solution.

|  |
| --- |
| var ClusteredEntityCollection = function (map, options) {  /\* Private Properties \*/  var \_map = map,  \_layer,  \_data = [],  \_numXCells,  \_numYCells;  //Base default options  var \_options = {  gridSize: 45,  clusterPlacementType: ClusterPlacementTypes.MeanAverage,  layerOffset: new Microsoft.Maps.Point(0, 0),  clusteringEnabled: true,  singlePinCallback: function (data) {  return new Microsoft.Maps.Pushpin(data.\_LatLong);  },  clusteredPinCallback: function (data) {  var pin = new Microsoft.Maps.Pushpin(data.\_LatLong, { text: '+' });  pin.description = cluster.length + " locations<br/>Zoom in for more details.";  return pin;  },  callback: null  };  /\* Private Methods \*/  function \_init() {  //Create an instance of an EntityCollection  \_layer = new Microsoft.Maps.EntityCollection();  \_map.entities.push(\_layer);  setOptions(options);  Microsoft.Maps.Events.addHandler(\_map, 'viewchangeend', cluster);  }  // Clusters the data and displays it on the map.  function cluster() {  //remove all pins from the layer  \_layer.clear();  if (\_data != null) {  //Clustering logic goes in here  }  //Call users callback  if (\_options.callback) {  \_options.callback();  }  }  //Updates the default options with new options  function setOptions(options) {  for (attrname in options) {  \_options[attrname] = options[attrname];  }  cluster();  }  /\* Public Methods \*/  /\*\*  \* @returns The clustering options.  \*/  this.GetOptions = function () {  return \_options;  };  /\*\*  \* Sets the clustering options.  \* Example: clusterLayer.SetOptions({ gridSize : 30});  \*/  this.SetOptions = function (options) {  setOptions(options);  };  /\*\*  \* Sets the data that is to be clustered and displayed on the map. All objects  \* must at minimium have a Latitude and Longitude properties.  \* The algorithm will convert them to a Location objects when loading in data.  \*/  this.SetData = function (data) {  if (data != null) {  \_data = data;  var i = \_data.length - 1;  if (i > 0) {  do {  \_data[i].\_LatLong = new Microsoft.Maps.Location(\_data[i].Latitude, \_data[i].Longitude);  } while (i--)  }  } else {  \_data = [];  }  cluster();  };  //Initialize class  \_init();  }; |

The original clustering algorithm created for v6 of Bing Maps had the logic to create the pushpins within the algorithm. This meant that if you wanted to modify this logic you had to modify the code for the algorithm. To make this new algorithm more flexible two callback functions can be set as options; *singlePinCallback*, and *clusteredPinCallback*. These callbacks are used to generate pushpins for the layer, thus separating the logic of creating the pushpins from the clustering algorithm. This makes it easy to have multiple clustered layers that all render data differently. The following is an example of how the options for creating pushpins can be created:

|  |
| --- |
| var options = {  singlePinCallback: function (data) {  return new Microsoft.Maps.Pushpin(data.\_LatLong);  },  clusteredPinCallback: function (data) {  var pin = new Microsoft.Maps.Pushpin(data.\_LatLong, { text: '+' });  pin.description = cluster.length + " locations<br/>Zoom in for more details.";  return pin;  }  }; |

Here is a list of all option properties that can be passed into the *ClusteredEntityCollection* class.

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| clusterPlacementType | int | The type of cluster placement method to use when representing a clustered location. |
| gridSize | int | The size of a grid cell in pixels. |
| clusteringEnabled | bool | A flag that indicates if clustering is enabled or not. All data that is in the map view is represented as a single pushpin. |
| layerOffset | Point | Offsets the layer by a specified amount of pixels. Only works when using the “center” cluster placement method. |
| singlePinCallback | function | A function that takes in a data object and returns a pushpin. |
| clusteredPinCallback | function | A function that takes in an array of data and a latlong coordinate then returns a clustered pushpin. |
| callback | function | A callback function that is called when the clustering algorithm has completed. This |

To increase the usability of this class a number of public methods have been added. The following is a list of all the available methods and a description of what they do.

|  |  |
| --- | --- |
| Name | Description |
| BringLayerToFront | Sets all layer's z-index to 0 and increases the z-index of the current layer to 1. Thus displaying the current layer on top. |
| GetData | Gets the data array and returns it to the user. |
| GetDisplayedData | Returns all data that is the current map view. |
| GetDataByGridKey | Returns an array of data that is in a particular grid cell. A grid key is used to determine this. As an option a boolean can be passed to indicate if the data is clustered or not. This would be known if retrieving the data based on a reference from its associated pin. |
| GetPinByGridKey | Returns a Pushpin that is in a grid cell. A grid key is used to determine this. |
| GetLayer | Returns a reference to the EntityCollection being used. |
| GetOptions | Returns the clustering options. |
| SetOptions | Sets the clustering options. |
| SetData | Sets the data that is to be clustered and displayed on the map. |

The following is an example of a basic implementation of the *ClusteredEntityCollection* class.

|  |
| --- |
| <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html>  <head>  <title>Client Side Clustering</title>  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">  <!-- Add a reference to the Bing Maps Control -->  <script charset="UTF-8" type="text/javascript" src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>    <script type="text/javascript">  var map, myLayer;  function GetMap() {  map = new Microsoft.Maps.Map(document.getElementById("myMap"),  { credentials: "Your\_Bing\_Maps\_Key" });  //Register and load the Client Side Clustering Module  Microsoft.Maps.registerModule("ClientSideClusteringModule", "scripts/V7ClientSideClustering.js");  Microsoft.Maps.loadModule("ClientSideClusteringModule", { callback: function () {  myLayer = new ClusteredEntityCollection(map, {  singlePinCallback: createPin,  clusteredPinCallback: createClusteredPin  });  });  }  function createPin(data) {  var pin = new Microsoft.Maps.Pushpin(data.\_LatLong);  //Add custom logic for creating single pushpin  return pin;  }  function createClusteredPin(cluster, latlong) {  var pin = new Microsoft.Maps.Pushpin(latlong, { text: '+' });  //Add custom logic for creating clustered pushpin  return pin;  }  //Makes a request for data  function RequestData() {  //Add logic to retrieve data from web service  }  function RequestDataCallback(response) {  if (response != null) {  myLayer.SetData(response);  }  }  </script>  </head>  <body onload="GetMap();">  <div id='myMap' style="position:relative; width:100%; height:500px;"></div>  </body>  </html> |

**Using Bing Maps Silverlight**

This article focused on the Bing Maps AJAX V7 control. However the algorithm used for client side clustering can easily be adopted to work with the Bing Maps Silverlight control.

**Source Code and Documentation**

Complete source code, with documentation and sample web pages is available [here](http://cid-0d35222484a76a01.office.live.com/embedicon.aspx/BingMapsv7/BMv7.ClientSideClustering.zip).