Places Near You with Nokia Maps and GeoNames API on Windows Phone

This article shows you the way to fetch famous places around you using GeoNames API. After getting the places information we will show them as markers/pushpins on maps using a Nokia Maps control.

Note: The WP7 version of this article can be accessed here.

Introduction

GeoNames is a large collection of database containing all countries information. This database contains more than eight million places names whose information can be freely downloaded.

Using GeoNames API, we can fetch many different types of information but in this article we will only fetch one type of information; *Places Information*. To fetch this information, we will use the Wikipedia API.

To draw pushpins on the Maps control, we will use the `Pushpin` control provided by the Windows Phone Toolkit.

Prerequisites

- Windows Phone development environment
- Registration at the GeoNames site
- Reference to Windows Phone Toolkit

References Required

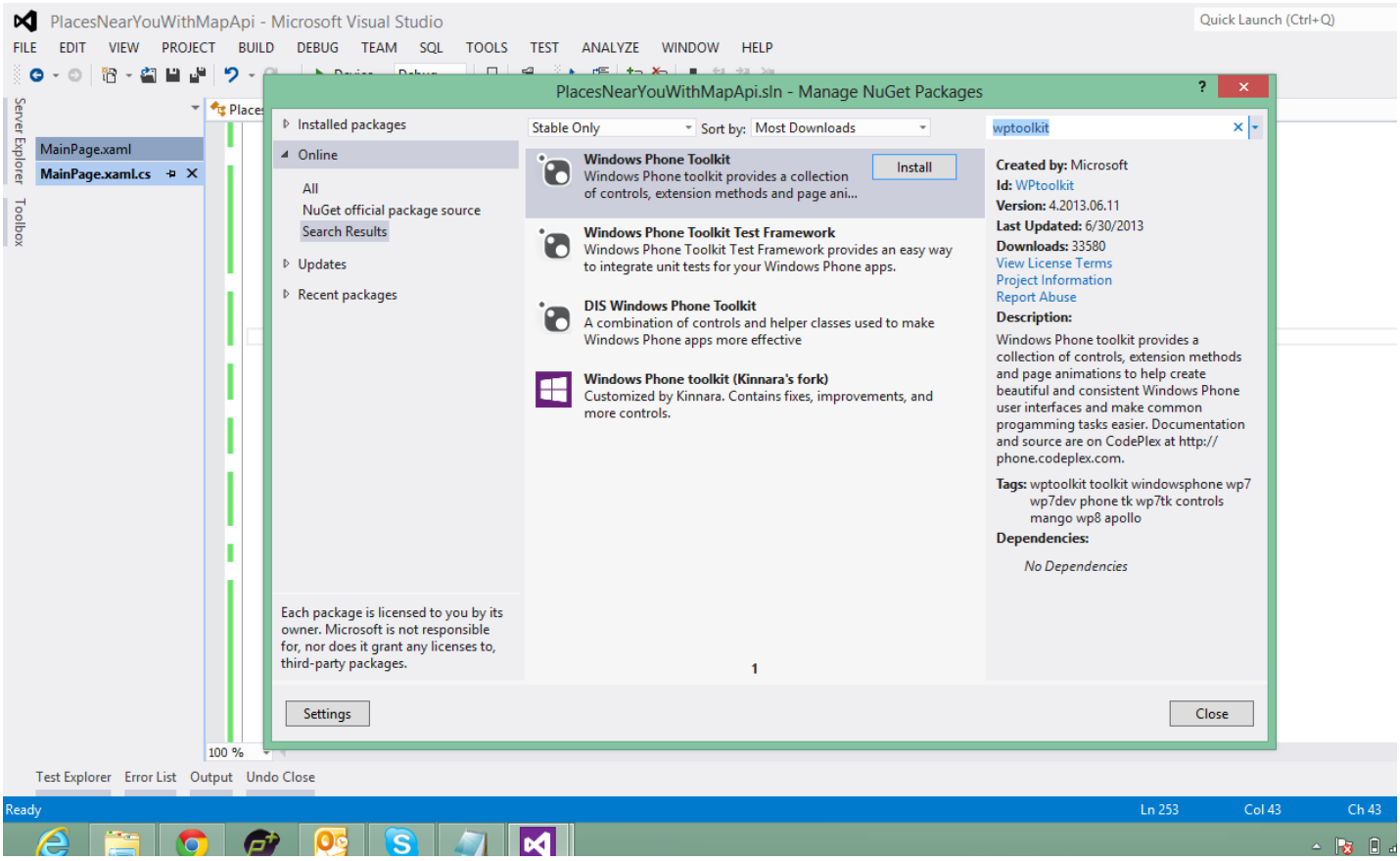
Following references to the DLLs are required.

- System.Collections.ObjectModel;
- System.IO.IsolatedStorage;
- Microsoft.Phone.Maps.Controls;
- Microsoft.Phone.Maps.Toolkit;

These can be added by **Project | References | Add Reference..**

Note: To add the Windows Phone Toolkit, search for the keyword "wptoolkit" from NuGet as shown in the following image.

NuGet is an inbuilt tool in VS2012.



Wikipedia API

To make use of Wikipedia API, we will have to make an HTTP server request at the link below:
http://api.geonames.org/findNearbyWikipediaJSON?
We will pass a few required parameters such as our own location, username etc. In response, we will get a list of near by places in JSON form.

Note: One of the parameters, i.e. **username**, is compulsory to send with each server request. To know more about

(C) Copyright Nokia 2014. All rights reserved.

username please check this link.

Creating UI

In our UI, we will make use of three text blocks and a Nokia maps control. First `TextBlock` will have the status message after fetching our device location. Second will have our own latitude value. Last `TextBlock` will have our longitude value. In our map control, we will show the resultant places with the help of pushpins.

The code in **MainPage.xaml** file is as shown below.

```
    <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="0" Margin="12,0,12,0">
            <Grid.RowDefinitions>
                <RowDefinition Height=".05*"/>
                <RowDefinition Height=".05*"/>
                <RowDefinition Height=".05*"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <TextBlock Grid.Row="0" HorizontalAlignment="Left"  x:Name="statusTextBlock"
Text="TextBlock" Visibility="Collapsed"/>
            <TextBlock Grid.Row="1" HorizontalAlignment="Left"
x:Name="latitudeTextBlock" Text="TextBlock" Visibility="Collapsed"/>
            <TextBlock Grid.Row="2" HorizontalAlignment="Left"
x:Name="longitudeTextBlock" Text="TextBlock" Visibility="Collapsed"/>
            <maps:Map Grid.Row="3" Name="mapControl" Visibility="Collapsed"
CartographicMode="Road">
                <toolkit:MapExtensions.Children>
                    <toolkit:MapItemsControl>
                        <toolkit:MapItemsControl.ItemTemplate>
                            <DataTemplate>
                                <toolkit:Pushpin GeoCoordinate="{Binding Coordinate}"
Content="{Binding Info}" />
                            </DataTemplate>
                        </toolkit:MapItemsControl.ItemTemplate>
                    </toolkit:MapItemsControl>
                </toolkit:MapExtensions.Children>
            </maps:Map>
        </Grid>
```

Initially, the visibility of the map control is set to `Collapsed` so that it can be shown when required. `MapExtensions` is an extension provided by the `Windows Phone Toolkit` to draw UI controls on a map. `MapExtensions` has a `Children` property which will hold the items to be shown on map. We are setting a `MapItemsControl` to the `Children` property which eventually will have lots of items to be displayed. While using the `Pushpin` control, we are making use of Binding method to bind the values of our geo-coordinates and information.

Code Behind

JSON Parsing

Following file(**Place.cs**) has been used for JSON parsing.

```
namespace PlacesNearYou.Data
{
    [DataContract]
    public class Place
    {
        [DataMember(Name = "summary")]
        public string Summary { get; set; }

        [DataMember(Name = "distance")]
        public string Distance { get; set; }

        [DataMember(Name = "rank")]
        public string Rank { get; set; }

        [DataMember(Name = "title")]
        public string Title { get; set; }

        [DataMember(Name = "wikipediaUrl")]
        public string WikipediaUrl { get; set; }

        [DataMember(Name = "elevation")]
        public string Elevation { get; set; }

        [DataMember(Name = "lng")]
```

 (C) Copyright Nokia 2014. All rights reserved.

```
        public string Longitude { get; set; }

        [DataMember(Name = "feature")]
        public string Feature { get; set; }

        [DataMember(Name = "lang")]
        public string Langauge { get; set; }

        [DataMember(Name = "lat")]
        public string Latitude { get; set; }
    }
}
```

Another file(**PlacesList.cs**) used for JSON parsing has the following declaration:

```
namespace PlacesNearYou.Data
{
    [DataContract]
    public class PlacesList
    {
        [DataMember(Name = "geonames")]
        public List<Place> PlaceList { get; set; }
    }
}
```

Creating structure for Binding

Following file(**PlaceToMap.cs**) has been used.

```
namespace PlaceToMap.Data
{
   public class PlaceToMap
   {
     public GeoCoordinate Coordinate { get; set; }
     public string Info { get; set; }
   }
}
```

Making Server Request

We will start with finding the current location of the phone. For that, we'll make use of `GeoCoordinateWatcher` API as explained in this article. After successfully fetching the current location of the device (in terms of latitude and longitude), we will start an HTTP request for the GeoNames API. The code snippet is shown below.

```
    private void startGeoNamesAPICall()
        {
            try
            {
                HttpWebRequest httpReq = (HttpWebRequest)HttpWebRequest.Create(new
Uri(AppConstants.baseUri + "&lat=" + currentLatitude + "&lng=" + currentLongitude +
"&username=" + AppConstants.strUserName + "&radius=" +
AppConstants.strDefaultRadiusForWikiAPI + "&maxRows=" +
AppConstants.strDefaultResultRowsForWikiAPI));
                httpReq.BeginGetResponse(HTTPWebRequestCallBack, httpReq);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

As shown above, we are passing a few parameters along with the base URL. These parameters are:

- Our own latitude
- Our longitude
- The user name (on behalf of which the request is made- **it is compulsory to provide a registered user name**). The username can be registered here. For more info on username, you may check this link.
- Radius - to determine in how much radius searching will be performed
- Maximum Rows - how many rows are required as max in the result

Parsing the response

The code snippet to handle response is shown below.

```
  private void HTTPWebRequestCallBack(IAsyncResult result)
```

 (C) Copyright Nokia 2014. All rights reserved.

```
        {
            string strResponse = "";

            try
            {
                Dispatcher.BeginInvoke(() =>
                {
                    try
                    {
                        HttpWebRequest httpRequest = (HttpWebRequest)result.AsyncState;
                        WebResponse response = httpRequest.EndGetResponse(result);
                        Stream stream = response.GetResponseStream();
                        StreamReader reader = new StreamReader(stream);
                        strResponse = reader.ReadToEnd();

                        parseResponseData(strResponse);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show(ex.Message);
                    }
                });
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
```

After converting the `WebResponse` type data into String, we need to parse it now as it is still in the JSON format. the code snippet for parsing is:

```
 private void parseResponseData(String aResponse)
        {
            placesListObj = new PlacesList();

            MemoryStream ms = new MemoryStream(Encoding.UTF8.GetBytes(aResponse));
            DataContractJsonSerializer ser = new
DataContractJsonSerializer(placesListObj.GetType());
            placesListObj = ser.ReadObject(ms) as PlacesList;
            ms.Close();

            // updating UI
            if (placesListObj != null)
            {
                updateMap(placesListObj);
            }
        }
```

Drawing pushpins on map

In case parsing is done successfully and we've got at least a single record in response, we will show its information on the map using Pushpin(s).

```
 private void updateMap(PlacesList aWiKIAPIResponse)
        {
            int totalRecords = aWiKIAPIResponse.PlaceList.Count();
            mapControl.Visibility = System.Windows.Visibility.Visible;

            try
            {
                ObservableCollection<PlaceToMap> placeToMapObjs = new
ObservableCollection<PlaceToMap>();
                for (int index = 0; index < totalRecords; index++)
                {
                    placeToMapObjs.Add(new PlaceToMap()
                    {
                        Coordinate = new
GeoCoordinate(Convert.ToDouble(aWiKIAPIResponse.PlaceList.ElementAt(index).Latitude),

Convert.ToDouble(aWiKIAPIResponse.PlaceList.ElementAt(index).Longitude)),
                        Info = aWiKIAPIResponse.PlaceList.ElementAt(index).Title +
Environment.NewLine + aWiKIAPIResponse.PlaceList.ElementAt(index).Feature
                    });
```

 (C) Copyright Nokia 2014. All rights reserved.

```
                }

                ObservableCollection<DependencyObject> children =
MapExtensions.GetChildren(mapControl);
                var obj = children.FirstOrDefault(x => x.GetType() ==
typeof(MapItemsControl)) as MapItemsControl;

                obj.ItemsSource = placeToMapObjs;
                mapControl.SetView(new GeoCoordinate(Convert.ToDouble(currentLatitude),
Convert.ToDouble(currentLongitude)), 14);
            }
            catch (Exception)
            {

            }
        }
```

We are mapping all the `Place` type objects into `PlaceToMap` type objects as we are using `Binding` method to bind the attribute values in the map control. After the conversion, we are getting a reference of the `Children` property of `MapExtensions` class. From the `children` object, we are finding the `first` or the `default` type, in our case it is `MapItemsControl`. Once we found its reference, we are setting it to point towards our `placeToMapObjs` object. Lastly, we are setting the map control's view as our current location and having a zoom level 14.

Build and Run

Now you may build the app and try to run it.

References

- geonames.org
- geonames.org Wikipedia Webservice
- Using Location Service APIs in WP7
- How to add UIElements to a Map control in Windows Phone 8
- Maps in Windows Phone 8 and Phone toolkit: a winning team – Part 2 (qmatteoq.com blog)

 (C) Copyright Nokia 2014. All rights reserved.