Create an Extension/Plugin for HydroDesktop to Import an Excel Worksheet

POSTED BY MUDNUG · APRIL 10, 2012
UPDATED BY ERIC HULLINGER AND CARLOS OSORIO-MURILLO - JULY 18, 2013

Introduction

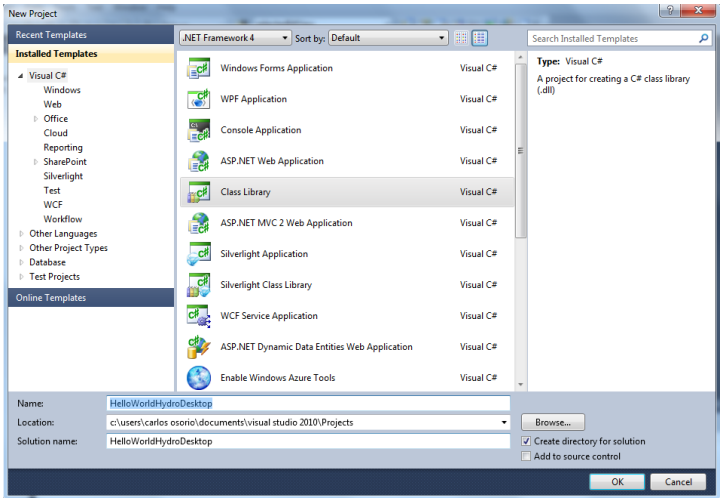
HydroDesktop uses DotSpatial GIS as an open-source project that contains controls which can be used to manipulate and display geographic information. This article explains how to create a DotSpatial extension. The extension we are creating will allow the user to import points from an Excel file. It will also demonstrate how to programmatically create a FeatureSet and add it to the map.

Downloading Visual Studio

There are several free IDEs available, namely SharpDevelop, MonoDevelop, and Visual Studio Express. None of these presently support the template, however, so you'll need Visual Studio Professional or better.

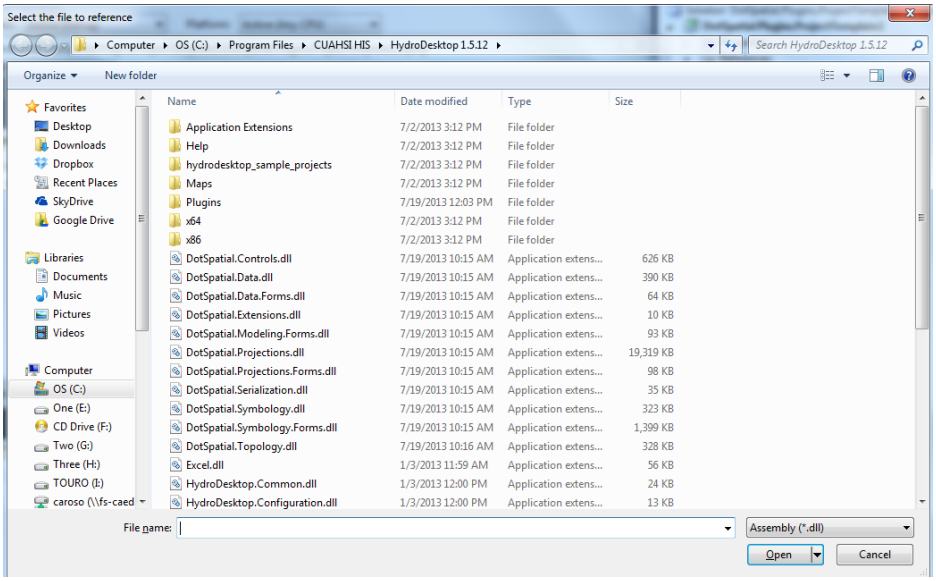
Creating a New Project

Start Visual Studio 2010 and open the New Project dialog (File, New, Project…). The type op project is Class library.



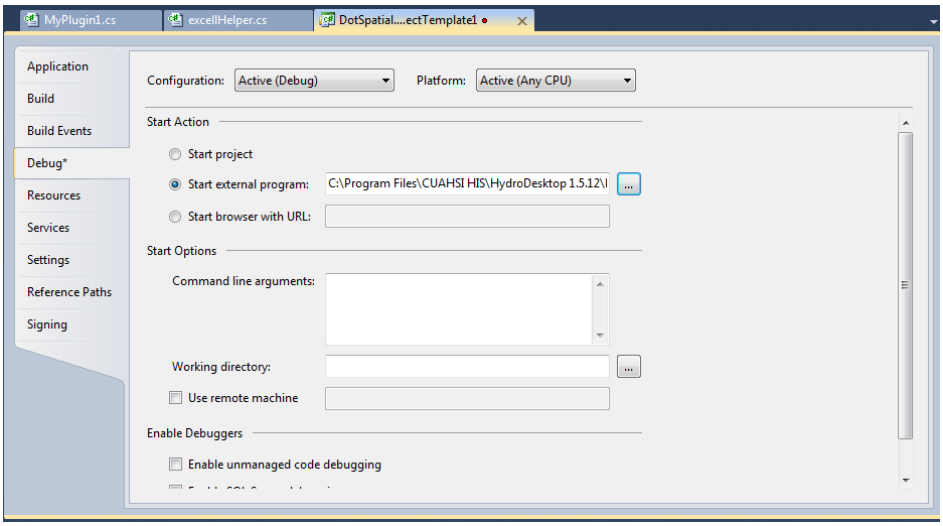
Add DotSpatial and HydroDesktop References

For this simple exercise you only need to add references to DotSpatial.Controls and DotSpatial.Extensions. However, you can safely add references to all the DotSpatial and HydroDesktop assemblies as shown below.

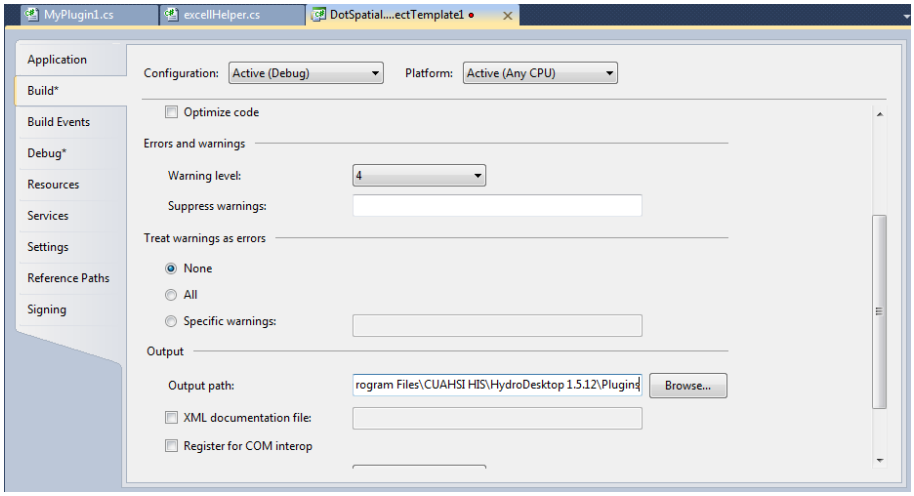


Set HydroDesktop as the Start Up Application

You can link the plugin with HydroDesktop changing the start program for the main executable of HydroDesktop. Please, go to *Properties, Debug*, and change the path of the *start external program* by the main executable of HydroDesktop located in the folder C:\Program Files\CUAHSI HIS\HydroDesktop 1.X.X.



It is needed to change also the build folder. Please, go to *Properties, Build*, and change *output path* by C:\Program Files\CUAHSI HIS\HydroDesktop 1.xxx\Plugins



Making an Text Extension

You may delete the Readme.txt and modify the name of the MyPlugin1 class (to reflect the functionality provided by the extension). I named mine ImportFromPlugin. Additionally, delete the current code found inside the ButtonClick event.

Add a new class file to the project (Project, Add New Item…) named **TextHelper**. We will place our logic in this class and call it from the button in our plugin class. This separation of concerns makes maintaining the code easier.

Using Windows Forms

We are going to prompt the user to select a file using the OpenFileDialog. This requires adding a reference to our project. Using the Add Reference dialog (Project, Add Reference…) find, and add a reference to System.Windows.Forms.

We also want to add a using statement to our project so that we can access the OpenFileDialog class without using its fully qualified namespace (System.Windows.Forms.OpenFileDialog) each time. This will also mean that we can use other classes in the same namespace, like the MessageBox class, without specifying the full namespace.

Add the statement beneath the other using statements, which are near the top of the file.

```
using System.Windows.Forms;
```

Getting the Selected File

Using the OpenFileDialog is fairly simple. We set the type of files we want the user to be able to select as the Filter. We then call ShowDialog() to freeze our extension while the user selects a file. If they don't click cancel, the FileName property will have the file they wish to open. Add this public method to your class, which will do this task.

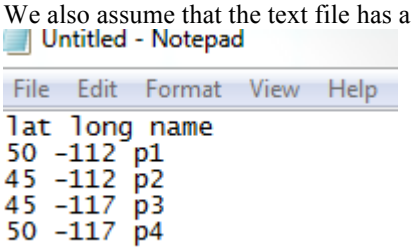
```
public static FeatureSet OpenExcelFile()
      {
         OpenFileDialog openDialog = new OpenFileDialog();
         openDialog.Filter = "Excel Files|*.xlsx";
         if (openDialog.ShowDialog() == DialogResult.OK)
         {
            DataTable excelTable = ConvertTextFileToDataTable(openDialog.FileName);
            return ConvertDataTableToFeatureSet(excelTable);
         }
         return null;
      }
```

We call two methods here that we haven't yet created. We'll create them next.

Converting the Text File to a DataTable

Working with files can pose a number of issues. For example, perhaps a process deletes the file after the user selects it and right before we open it. We'll ignore most of these cases to keep things simple.

We also assume that the text file has a header with a lat and a long column.



You'll want to add a few more using statements to your class file.

```
using System.Data;
using System.IO
using DotSpatial.Projections;
using DotSpatial.Data;
using DotSpatial.Topology;
```

And the following static method.

```
private static DataTable ConvertTextFileToDataTable(string excelFileName)
        {
            string[] txt = File.ReadAllLines(excelFileName);

            DataTable table = new DataTable();
            table.Columns.Add("lat", typeof(double));
            table.Columns.Add("long", typeof(double));
            table.Columns.Add("name", typeof(string));

            for (int i = 1; i < txt.Length; i++)
            {
                double lat= Convert.ToDouble(txt[i].Split(' ')[0]);
                double lng= Convert.ToDouble(txt[i].Split(' ')[1]);
                string name= Convert.ToString(txt[i].Split(' ')[2]);
                table.Rows.Add(lat, lng, name);
            }
```

```
            return table;

        }
```

You'll notice that we've used static methods, because our code follows a functional style and there aren't any objects to which our class needs to hold a reference.

Converting the DataTable to a FeatureSet

Creating a FeatureSet from a DataTable is fairly straightforward. I'll let you examine the commented code below.

```
private static FeatureSet ConvertDataTableToFeatureSet(DataTable txtTable)
    {
      // See if table has the lat, long columns
      if (txtTable.Columns.Contains("lat") & txtTable.Columns.Contains("long"))
      {
        FeatureSet fs = new FeatureSet(FeatureType.Point);
        fs.DataTable.Columns.Add("Name");
        DotSpatial.Projections.GeographicCategories.World world = new Projections.GeographicCategories.World();
        fs.Projection = world.WGS1984;
        //; KnownCoordinateSystems.Geographic.World.WGS1984;
        int i=0;
        foreach (DataRow txtRow in txtTable.Rows)
        {
          double lat = Double.Parse(txtRow["lat"].ToString());
          double lon = Double.Parse(txtRow["long"].ToString());

          // Add the point to the FeatureSet
          Coordinate coord = new Coordinate(lon, lat);
          Point point = new Point(coord);
          IFeature feature = fs.AddFeature(point);

            feature.DataRow["Name"] = txtRow[2];
            i++;

        }
        return fs;
```

```
      }
      else
      {
        MessageBox.Show("The excel table must have lat and long columns.");
        return null;
      }
    }
```

Wire up the Plugin to access our helper class

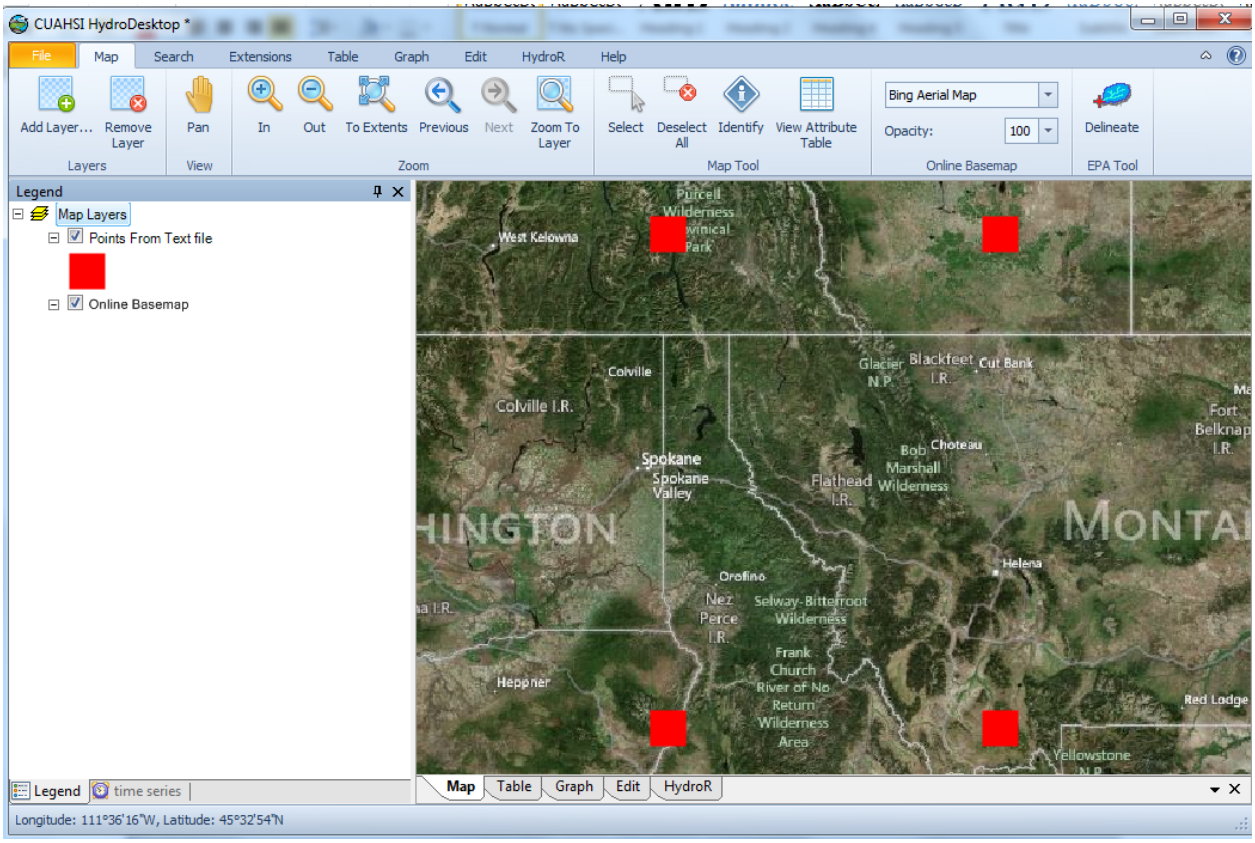
Our helper class is complete and we can reference it from our plugin class. Open ImportFromTextPlugin. Change the name of the SimpleActionItem from “My Button Caption” to “Add Layer from Excel…”

Replace the ButtonClick event handler. Here, we add the FeatureSet created by our TextHelper to the map and assign it LegendText.

```
public void ButtonClick(object sender, EventArgs e)
{
   var featureSet = TextHelper.OpenExcelFile();

   if (featureSet != null)
   {
      //add feature set to map
      var layer = App.Map.Layers.Add(featureSet);
      layer.LegendText = "Points From Text";
   }
}
```

We are done! Build and run your project.


CUAHSI HydroDesktop *
File
Map
Search
Extensions
Table
Graph
Edit
HydroR
Help
Add Layer...
Remove Layer
Layers
Pan
View
In
Out
To Extents
Previous
Next
Zoom To Layer
Zoom
Select
Deselect All
Identify
View Attribute Table
Map Tool
Bing Aerial Map
Opacity:
100
Online Basemap
Delineate
EPA Tool
Legend
Map Layers
Points From Text file
Online Basemap
Purcell Wilderness
West Kelowna
Colville
Colville I.R.
Glacier N.P.
Blackfeet I.R.
Cut Bank
Fort Belknap I.R.
Spokane
Spokane Valley
Bob Marshall Wilderness
Choteau
Flathead I.R.
Helena
Orofino
Nez Perce I.R.
Selway-Bitterroot Wilderness
Frank Church River of No Return Wilderness Area
Hepponer
Red Lodge
Yellowstone
Legend
time series
Map
Table
Graph
Edit
HydroR
Longitude: 111°36'16"W, Latitude: 45°32'54"N