Create a custom plugin DotSpatial.

Tutorial (8)

Purpose of this tutorial: Designing a custom plug-in in DotSpatial.

For example the following plug-in is used to get points form an Excel worksheet and plot them on the map.

There is a template available for creating a DotSpatial plugin. However, this template only works with Visual Studio 2010. For this tutorial we will start from scratch. However, creating a plugin without the template is doable.

**Step 1:**Create a new project in Visual Studio Environment. Instead of creating a windows form scroll down until you find the Class Library Template.

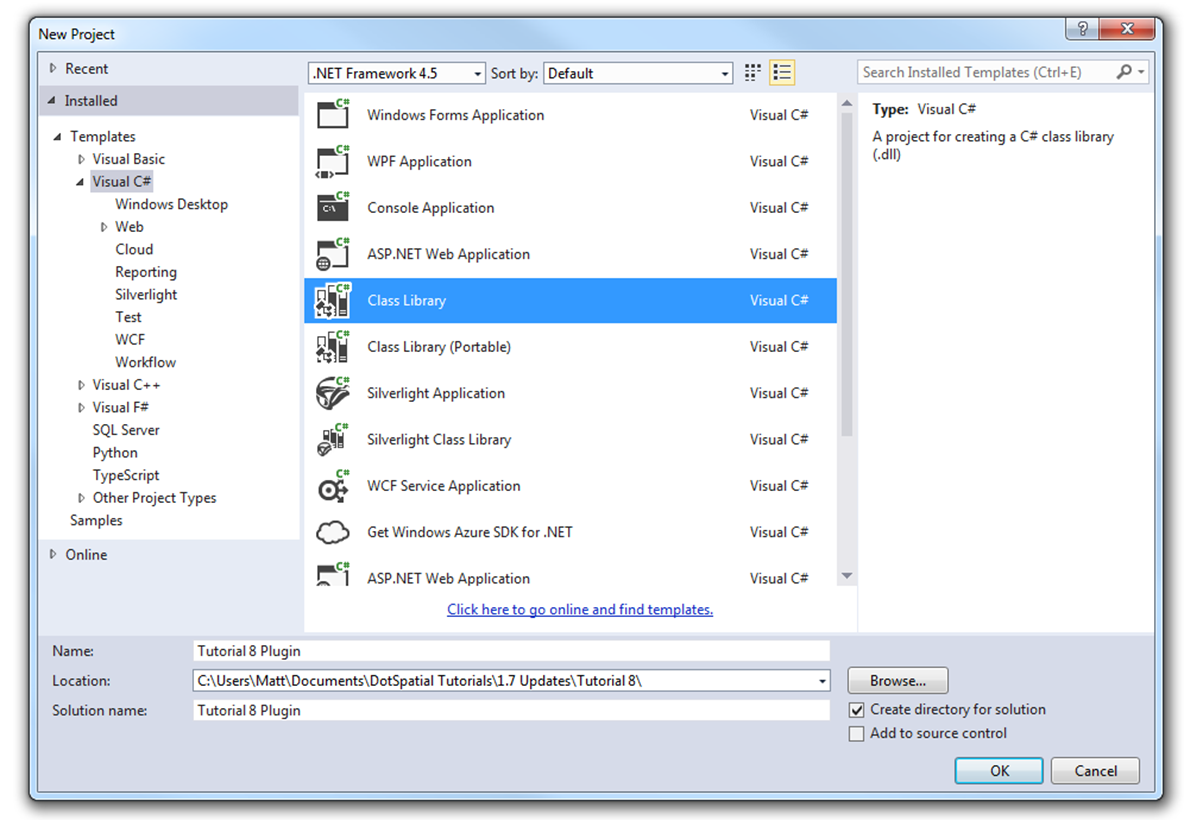
**

Figure 1: Class Library

**Step 2:** Double click on the Class Library Template to create the plug-in project.

**Step 3:** Get the DotSpatial dlls from <http://dotspatial.codeplex.com>

**Step 4 (Optional):** Link the plugin with your DotSpatial project. If you want to link the plugin to your project and test it directly you can link the two together.

Right click on the project in the solution explorer and choose Properties.

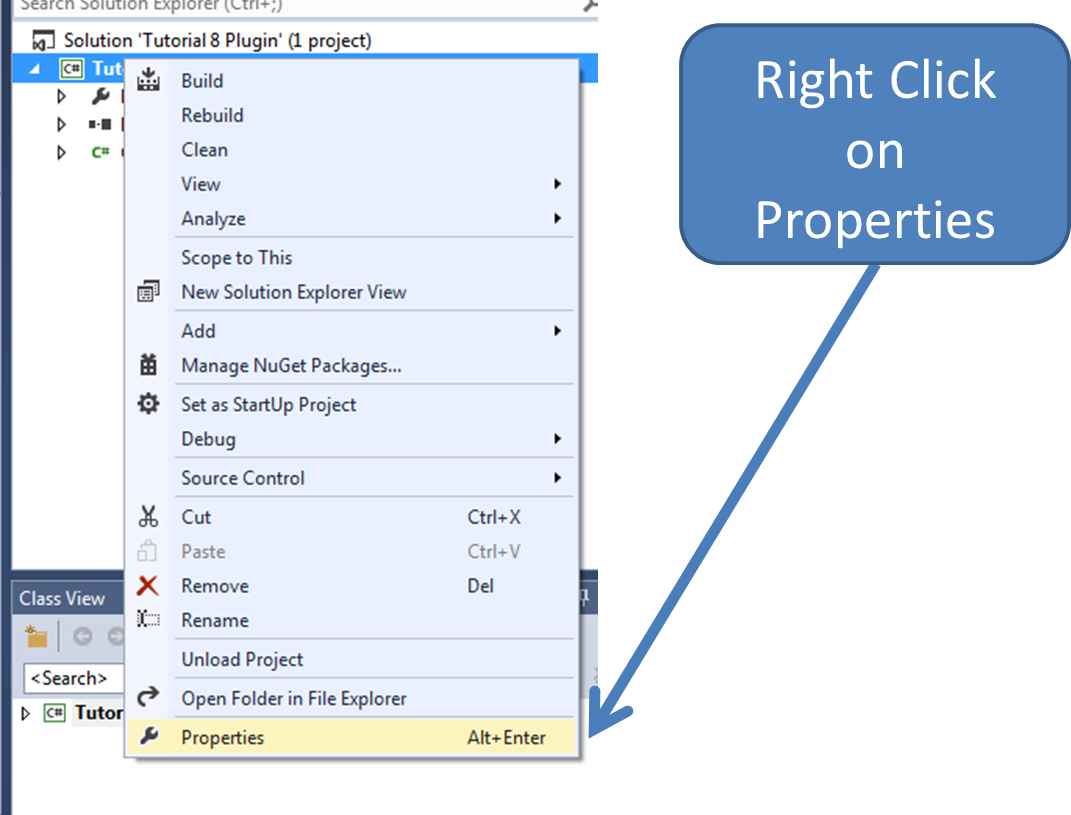


Figure : Properties

Navigate to the Build tab and scroll down to the Output Path.

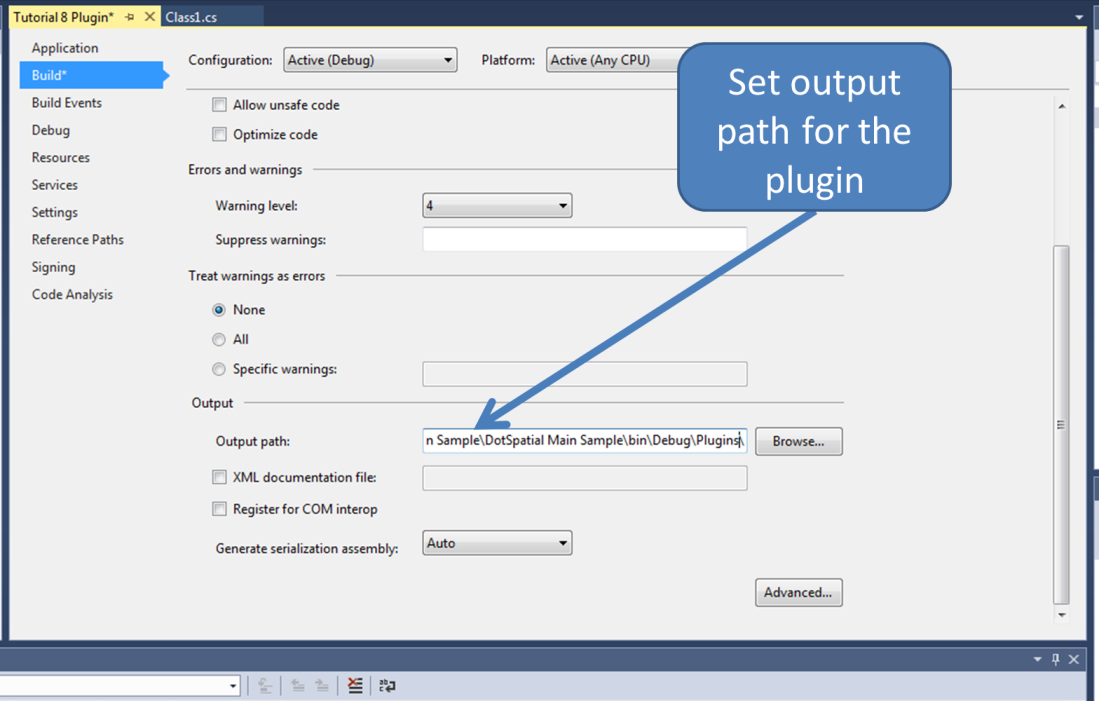


Figure : Output Path

The file path for this option should be to your main project’s Plugins folder. Once you build the project it will put the dll from your plugin project to the plugins folder in your main project, thus saving you from having to copy and paste the dll manually.

Next go to the debug tab and set the project to start an external program. Set the file path to your main projects .exe file.

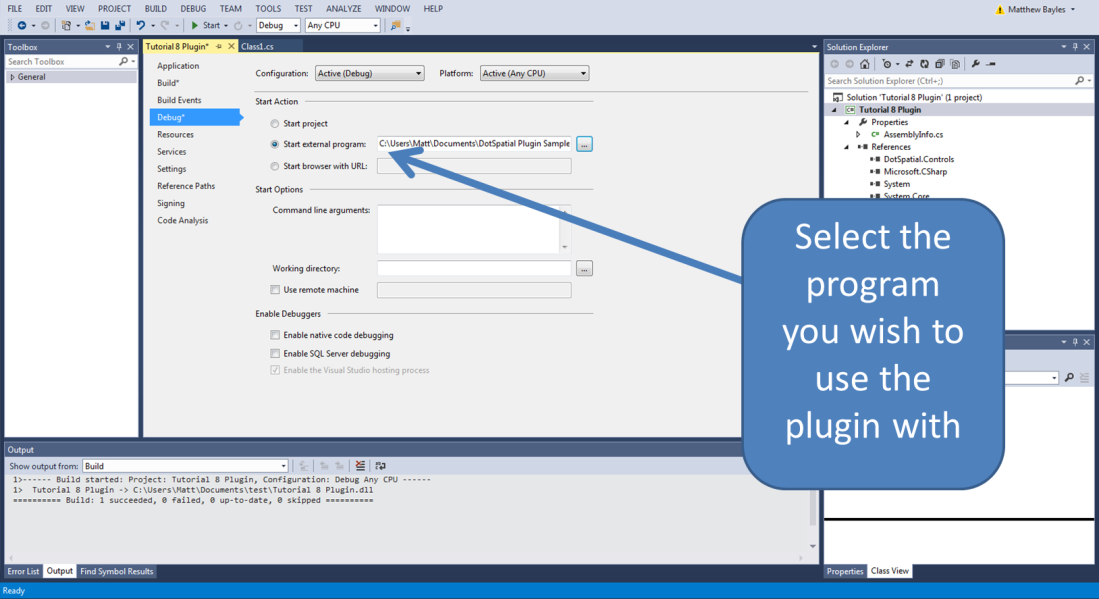


Figure : Starting External Program

When you run your project it will now start your main project with your new plugin loaded, thus making it possible to easily test the effectiveness of your new plugin.

**Step 5:** Code implementation.

Rename **Class1.cs** to **ExcelPlugin.cs**. Write the following code in the **ExcelPlugin.cs** class.

Add the following statements:

using System.Data;

using System.Data.OleDb;

using DotSpatial.Data;

using DotSpatial.Topology;

using DotSpatial.Projections;

using DotSpatial.Controls;

using DotSpatial.Extensions;

using DotSpatial.Controls.Header;

using System.Windows.Forms;

On the same line as your public class declaration add :Extension.

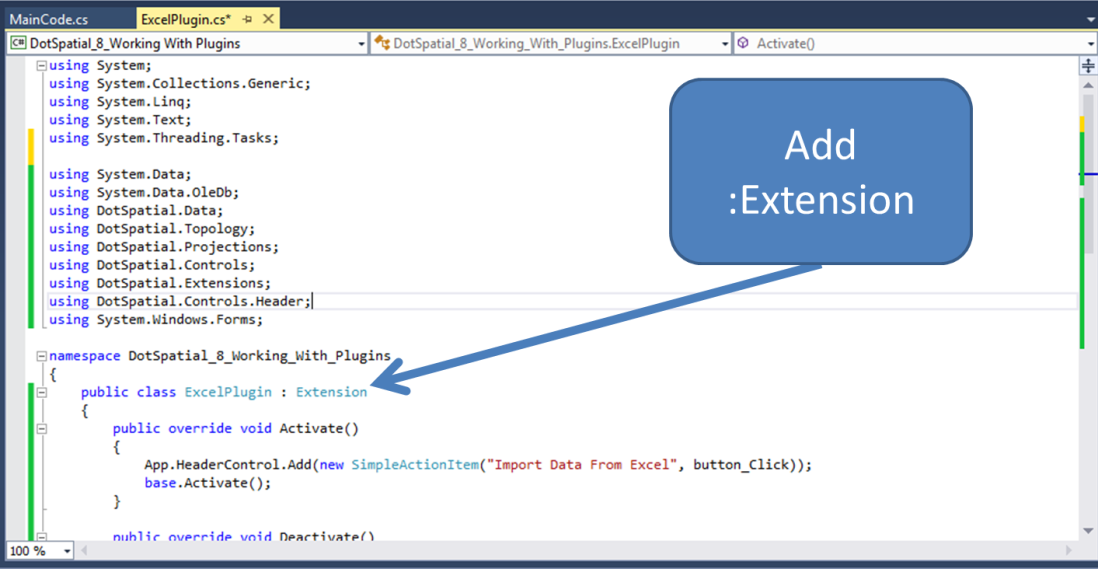


Figure : Add :Extension

public override void Activate()

{

App.HeaderControl.Add(new SimpleActionItem("Import Data From Excel ", ButtonClick));

base.Activate();

}

public override void Deactivate()

{

App.HeaderControl.RemoveAll();

base.Deactivate();

}

public void ButtonClick(object sender, EventArgs e)

{

var featureSet = MainCode.OpenExcelFile();

if (featureSet != null)

{

//add feature set to map

var layer = App.Map.Layers.Add(featureSet);

layer.LegendText = "Points From Excel";

}

}

Create a new class by going to project and selecting Add Class…

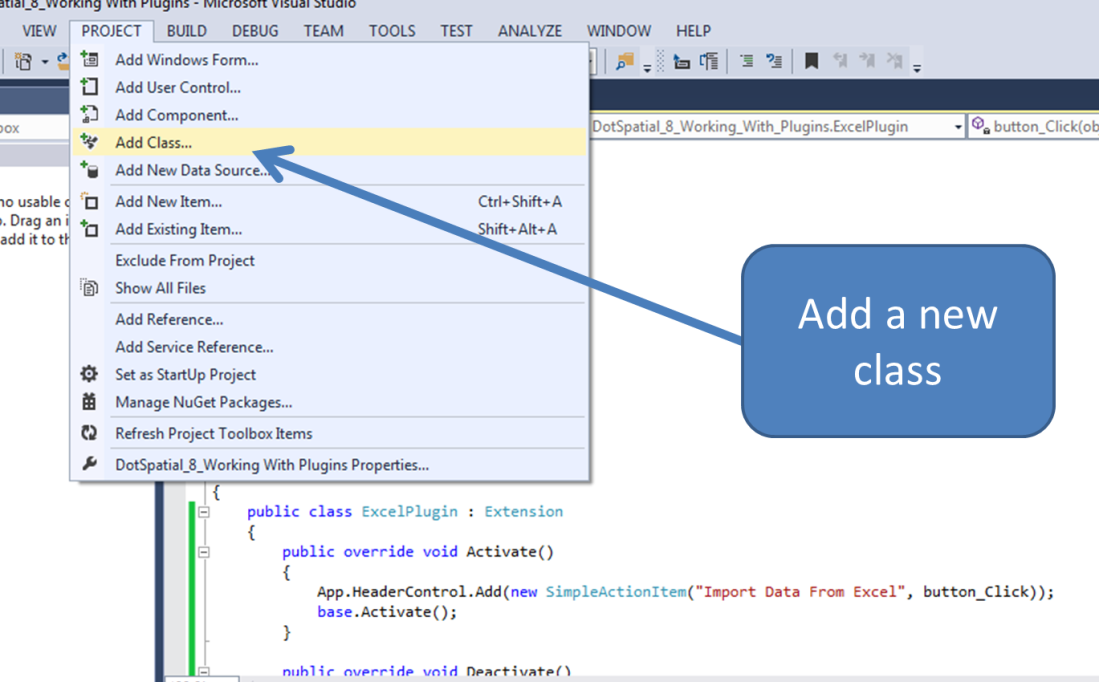


Figure : Adding a New Class

Name the new class **MainCode.cs**.

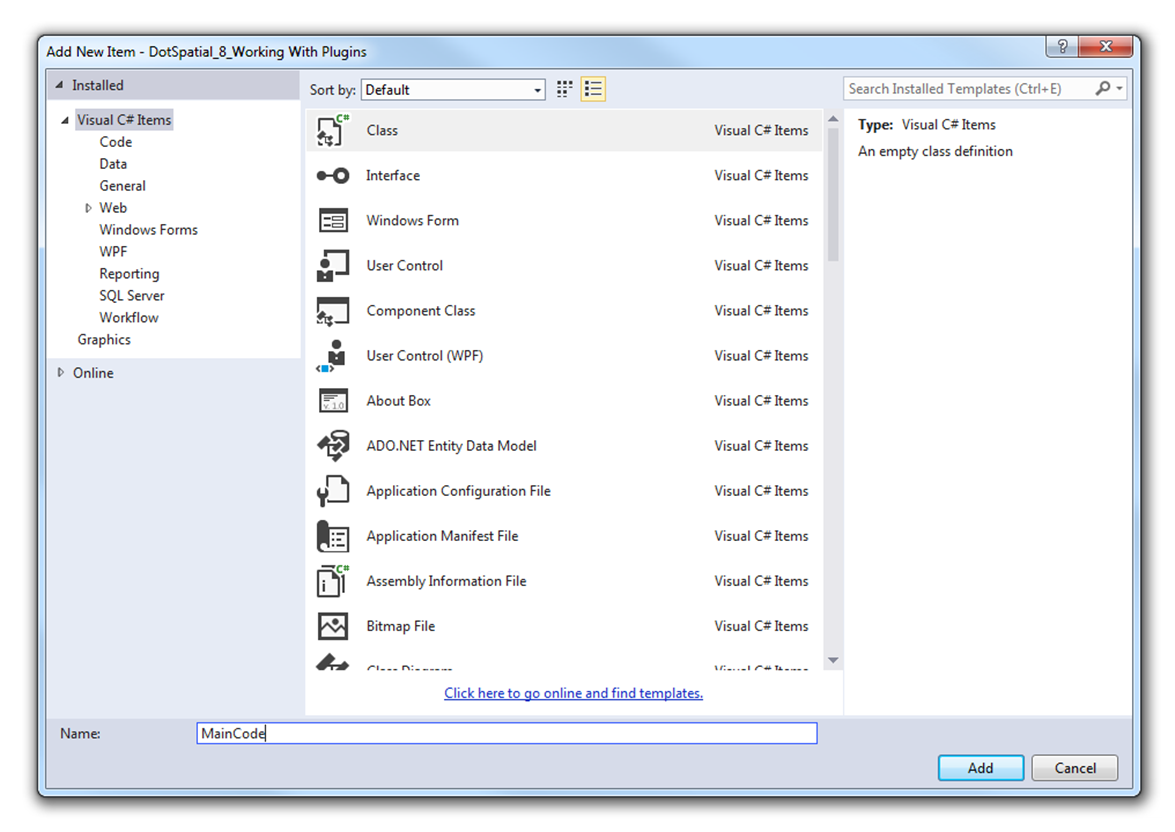


Figure : MainCode.cs

Add the following statements to **MainCode.cs**:

using System.Data;

using System.Data.OleDb;

using DotSpatial.Data;

using DotSpatial.Topology;

using DotSpatial.Projections;

using DotSpatial.Controls;

using DotSpatial.Extensions;

using DotSpatial.Controls.Header;

using System.Windows.Forms;

Next add the following three methods to this class:

public static FeatureSet OpenExcelFile()

{

OpenFileDialog openDialog = new OpenFileDialog();

openDialog.Filter = "Excel Files|\*.xlsx";

if (openDialog.ShowDialog() == DialogResult.OK)

{

DataTable excelTable = ConvertExcelFileToDataTable(openDialog.FileName);

return ConvertDataTableToFeatureSet(excelTable);

}

return null;

}

private static DataTable ConvertExcelFileToDataTable(string excelFileName)

{

string connectionString =

String.Format("Provider=Microsoft.ACE.OLEDB.12.0;Data Source={0};Extended Properties=\"Excel 12.0 Xml;HDR=YES;IMEX=1\"", excelFileName);

using (OleDbConnection connection = new OleDbConnection(connectionString))

{

string query = "SELECT \* FROM [Sheet1$]";

connection.Open();

OleDbCommand command = new OleDbCommand(query, connection);

OleDbDataAdapter adapter = new OleDbDataAdapter(command);

DataTable excelTable = new DataTable();

dapter.Fill(excelTable);

return excelTable;

}

}

private static FeatureSet ConvertDataTableToFeatureSet(DataTable excelTable)

{

// See if table has the lat, long columns

if (excelTable.Columns.Contains("lat") & excelTable.Columns.Contains("long"))

{

FeatureSet fs = new FeatureSet(FeatureType.Point);

fs.Projection = KnownCoordinateSystems.Geographic.World.WGS1984;

// Set columns of attribute table

fs.DataTable = excelTable.Clone();

foreach (DataRow excelRow in excelTable.Rows)

{

double lat = Double.Parse(excelRow["lat"].ToString());

double lon = Double.Parse(excelRow["long"].ToString());

// Add the point to the FeatureSet

Coordinate coord = new Coordinate(lon, lat);

Point point = new Point(coord);

IFeature feature = fs.AddFeature(point);

// Bring over all of the data as attribute data.

for (int i = 0; i <= excelTable.Columns.Count - 1; i++)

{

feature.DataRow[i] = excelRow[i];

}

}

return fs;

}

else

{

MessageBox.Show("The excel table must have lat and long columns.");

return null;

}

}

**Step 6:** Create an Excel Worksheet as follows:

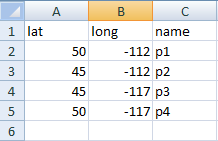


Figure : Excel Worksheet

**Step 7:** Output screen shot.

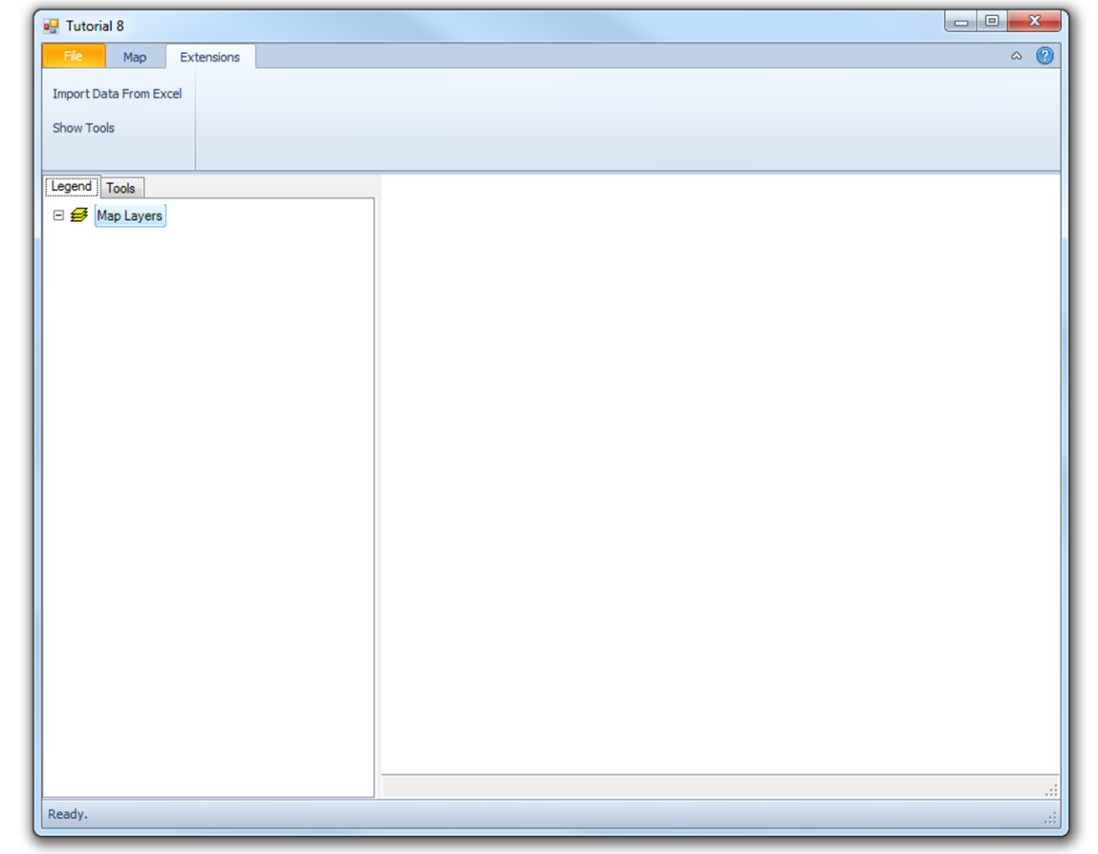


Figure : Run the Excel Plugin

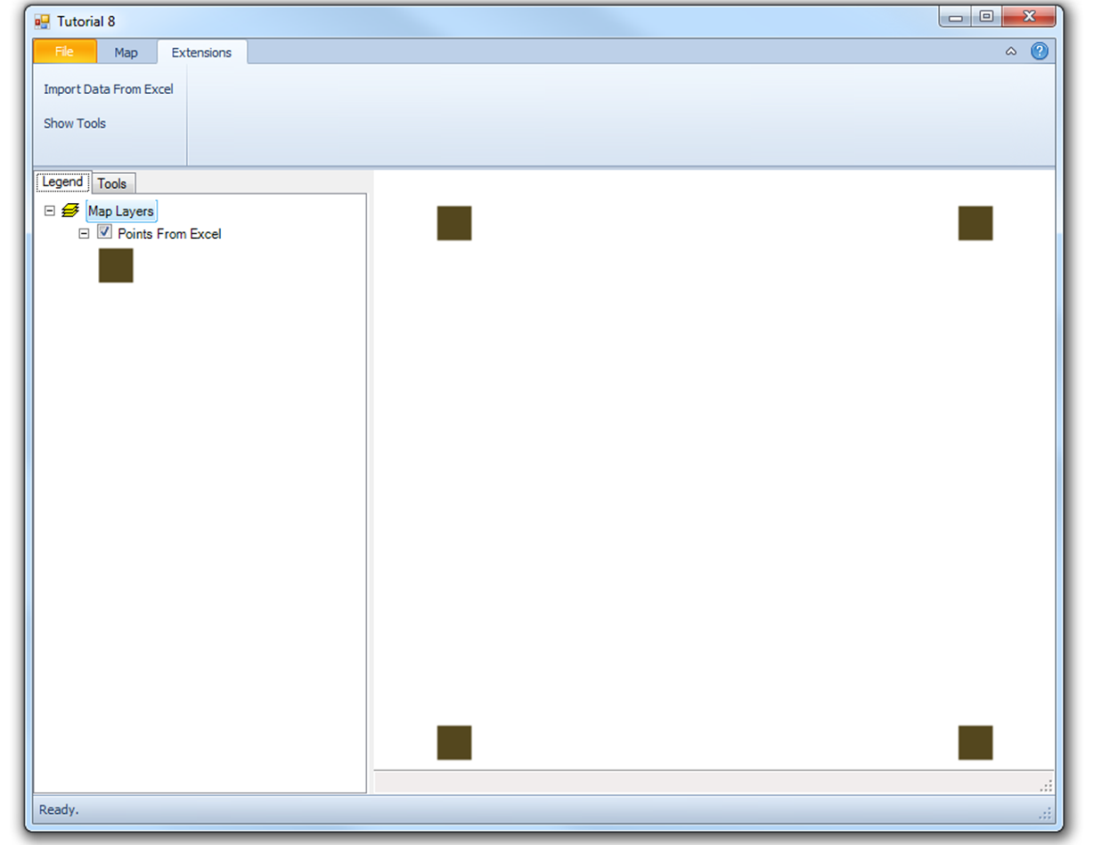


Figure : Final Output

Create a menu item for your plugin

Write the following code in the MyPlugin1.cs class

public override void Activate()

{

App.HeaderControl.Add(new RootItem("test\_root\_key", "Test Menu Item"));

App.HeaderControl.Add(new SimpleActionItem("test\_root\_key"," Import Data From Excel", ButtonClick));

base.Activate();

}

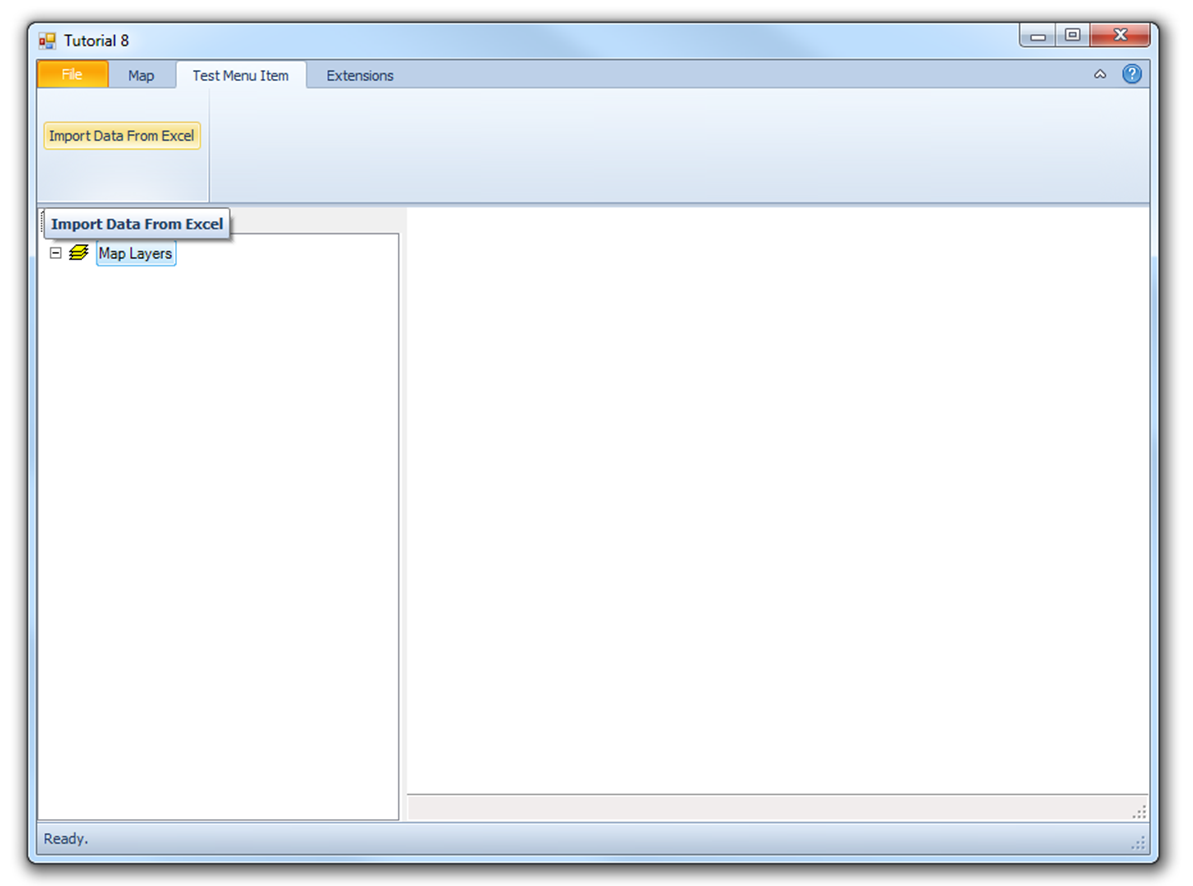


Figure : Create a Menu Item for the Plugin