

Developing Plugins for HydroDesktop in c#: Quick Start Tutorial

**A guide to help you get started developing plugins for HydroDesktop**

June 17, 2011

by:

Jiří Kadlec

Geospatial Software Lab

Idaho State University

Contents

[Introduction 2](#_Toc296394128)

[System Requirements 2](#_Toc296394129)

[How HydroDesktop Works: 3](#_Toc296394130)

[HydroDesktop SQLite Database: 3](#_Toc296394131)

[Creating a new HydroDesktop Plugin 5](#_Toc296394132)

[Step 1 – Create a new visual studio project 6](#_Toc296394133)

[Step 2 – Add References to DotSpatial and HydroDesktop 7](#_Toc296394134)

[Step 3 – Implement the IMapPlugin interface 9](#_Toc296394135)

[Step 4 – Add Ribbon Toolbar Button 11](#_Toc296394136)

[Step 4 – implement the plugin functionality 12](#_Toc296394137)

[Additional programming tasks 18](#_Toc296394138)

[Appendix: IMapPluginArgs Properties 19](#_Toc296394139)

[Map 19](#_Toc296394140)

[Legend 20](#_Toc296394141)

[Ribbon 20](#_Toc296394142)

[ProgressHandler 20](#_Toc296394143)

[Working With HydroDesktop Database 21](#_Toc296394144)

[HydroDesktop.Database.DbOperations class 21](#_Toc296394145)

[HydroDesktop.Database.RepositoryManagerSQL Class 22](#_Toc296394146)

# Introduction

HydroDesktop is a free and open source Geographic Information Systems (GIS) application that helps you discover, use, and manage hydrologic time series data. The functionality of HydroDesktop can be extended by plugins. This guide explains the main elements of the plugin interface, shows how to create a HydroDesktop plugin, and demonstrates how to access the HydroDesktop data repository SQLite database from within a plugin.

# System Requirements

Developing plugins for HydroDesktop requires the following components to be installed on the system:

* **Microsoft .NET Framework 4.0**
* **Microsoft Visual Studio** **2010** (Express versions are freely available)
* Latest release of **HydroDesktop**. (can be downloaded at http://www.hydrodesktop.org)
* User should be familiar with the Microsoft .NET framework and programming in C# .NET.

# How HydroDesktop Works:

The HydroDesktop software system consists of 4 major components:

1. HydroDesktop main application
2. HydroDesktop components
3. DotSpatial map components
4. SQLite Data Repository database
5. Default plug-ins

The HydroDesktop main application contains the main window and ribbon toolbar. The HydroDesktop components are software libraries (assemblies) with functions for querying the CUAHSI web services and accessing the local data repository database. The DotSpatial map components include the main map and legend. These components provide the HydroDesktop GIS (geographic information system) functionality. The SQLite data repository database stores the time series including all metadata downloaded by the user. For each project a different database can be used. Most of the functionality of HydroDesktop is provided in the form of plugins. The default plugins shipped with HydroDesktop include: Edit View, Graph View, Table View, HydroR, HydroModeler, EPA Delineation, Data Export, Basemap Fetcher, Metadata Fetcher and Search.

# HydroDesktop SQLite Database:

For writing a HydroDesktop plug-in that accesses or modifies hydrological time series data, it is important to understand the HydroDesktop data repository database. This database has a large number of tables so that both water quantity and water quality time series can be described with complete metadata as approved by the CUAHSI organization.

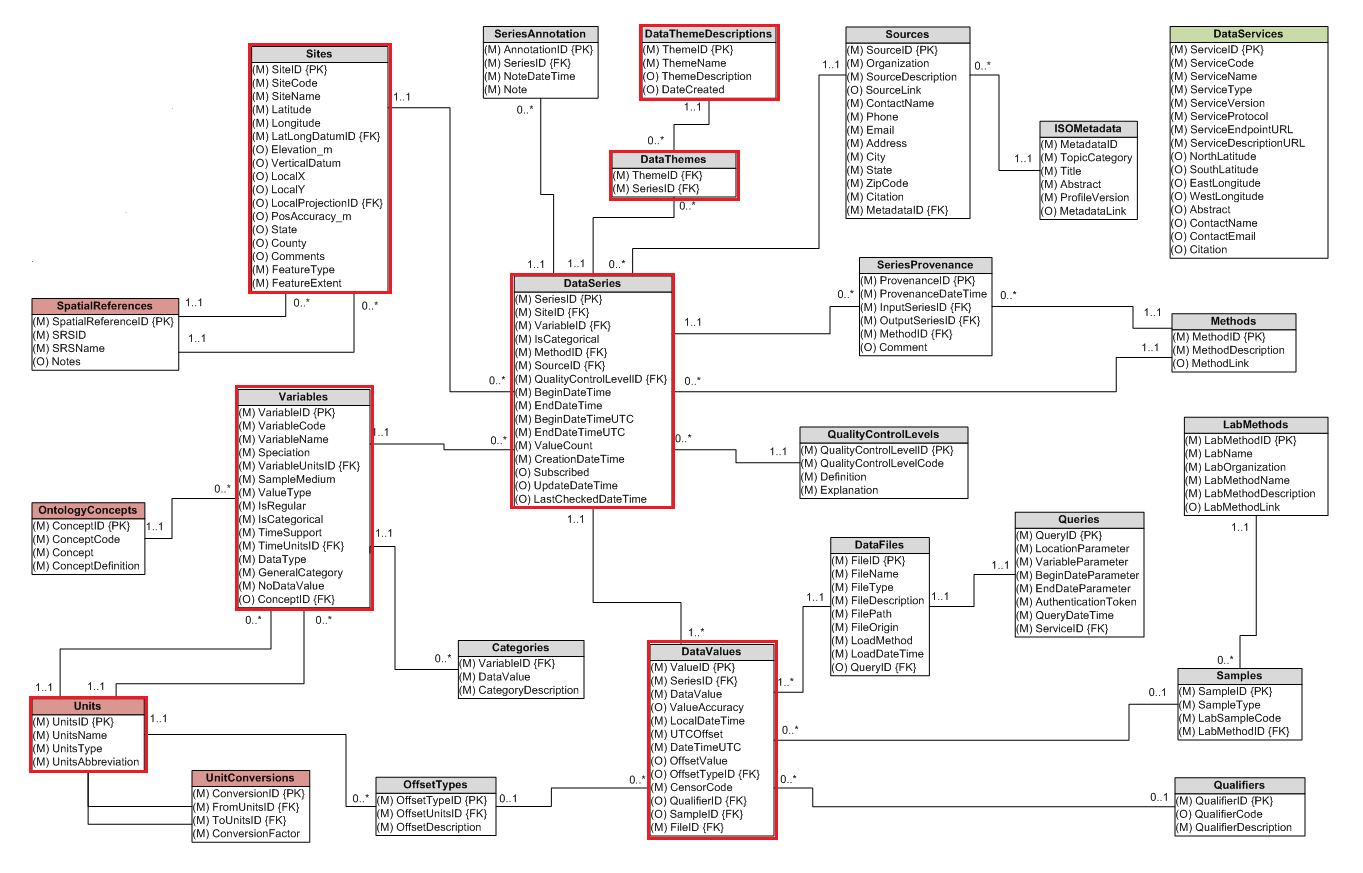


Figure HydroDesktop Database schema. The main tables described in detail are highlighted in red color.

Most important tables in the database are:

Table Main tables of the HydroDesktop database

|  |  |
| --- | --- |
| Database table | Description: |
| Sites | Information about the sites (locations) associated with the time series. Each site has a unique code, and latitude /longitude coordinates. There may be one or more time series associated with each site. |
| Variables | Information about the variables associated with the time series. Each variable has an unique code, name and units. An example of a variable is: *discharge*, *snow depth (cm)*, *precipitation amount (mm)* |
| Units | The variable units and time units corresponding to each variable |
| DataValues | This table contains the actual time series data. Each data value has the local time, UTC time and value and is a part of one data series. The SeriesID foreign key is the link between DataSeries and DataValues. Each DataSeries can contain one or many data values that are ordered by time. |
| DataSeries | The DataSeries table contains metadata about each time series. It defines the grouping of DataValues based on Site, Variable, Method, Source and QualityControlLevel. The DataSeries also has a StartTime, EndTime and ValueCount. |
| DataThemeDescriptions | This table contains the names and descriptions of data themes. A data theme is a grouping of time series (for example: *snow cover in Idaho*, *Texas coast water quality..*). Each theme is shown as a separate layer in the HydroDesktop map |
| DataThemes | This is the link table between DataSeries and DataTheme Descriptions. One series can be a part of one or more themes (relationship between Themes and DataSeries is a many-to-many relationship) |

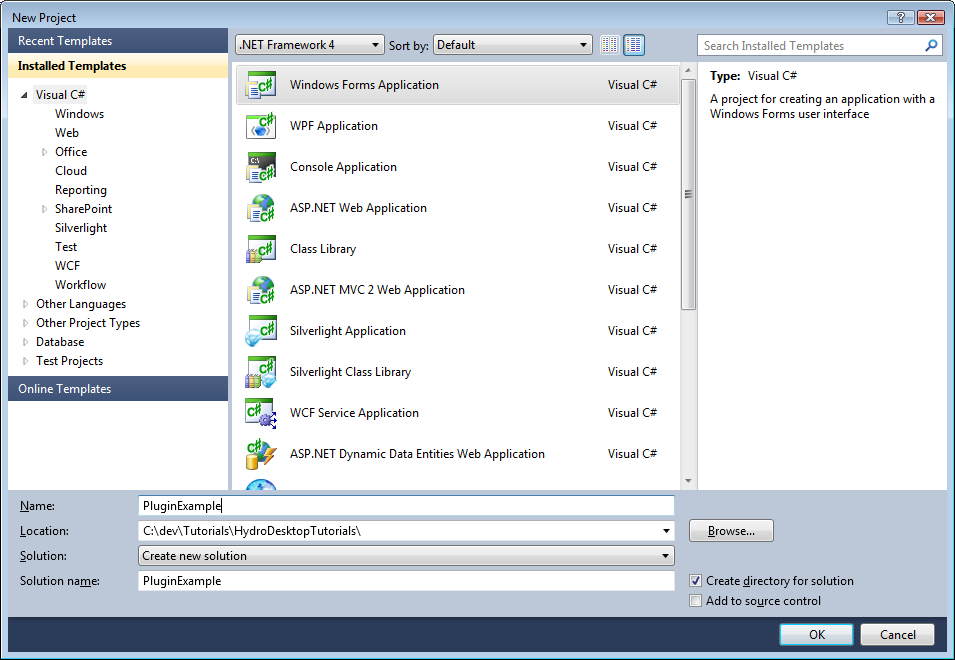
# Creating a new HydroDesktop Plugin

Creating a new HydroDesktop plugin has following steps:

1. Create a new Visual Studio Project
2. Add references to DotSpatial and HydroDesktop
3. Implement the **IMapPlugin interface**
4. Design the GUI – add items to HydroDesktop ribbon toolbar
5. Implement the plugin functionality

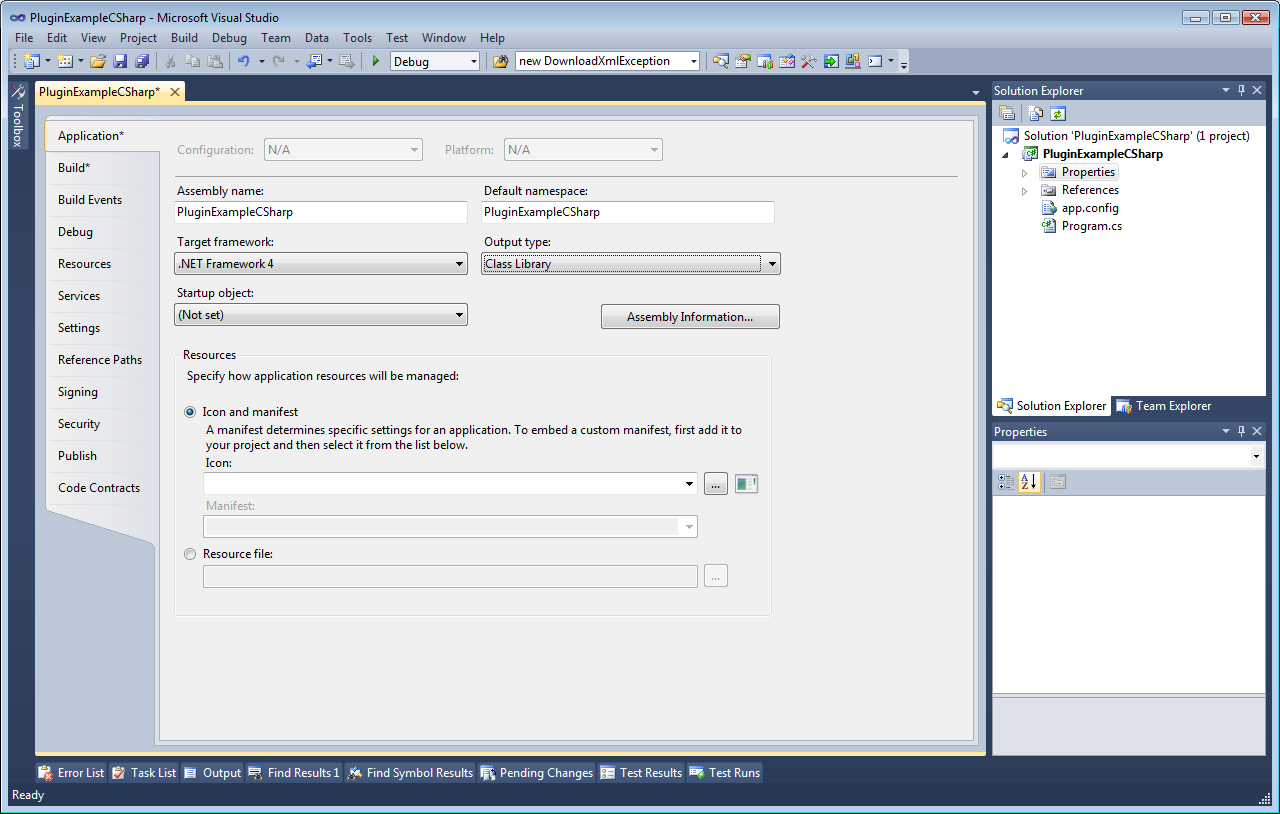
Step 1 – Create a new visual studio project

Open Visual Studio and select File – New – Project. Select project type **Class Library**. Specify the location of the project: PluginExample.



After the new project is created, several properties must be set.

1. In solution explorer, double-click on Properties. Change the target framework from .NET Framework 4 Client Profile to **.NET Framework 4**.



1. Go to the Build tab. Change the Output path from bin\Debug to:

**C:\Program Files\CUAHSI HIS\HydroDesktop\Plugins\**

(*Note: use the folder where HydroDesktop is installed. On some computers this may be on a different drive than the c: drive*)

Step 2 – Add References to DotSpatial and HydroDesktop

1. In solution explorer, right-click on References and choose Add Reference. Choose Browse and browse to the folder where HydroDesktop is installed (c:\Program Files\CUAHSI HIS\HydroDesktop.) Add references to the following assemblies:
   * DotSpatial.Data.dll
   * DotSpatial.Projections.dll
   * DotSpatial.Symbology.dll
   * DotSpatial.Symbology.Forms.dll
   * DotSpatial.Topology.dll
   * HydroDesktop.Configuration.dll
   * HydroDesktop.Data.dll
   * HydroDesktop.Interfaces.dll
2. Add reference to **System.Windows.Forms**. This reference must be added by clicking on the .NET tab in the Add Reference dialog and choosing System.Windows.Forms.

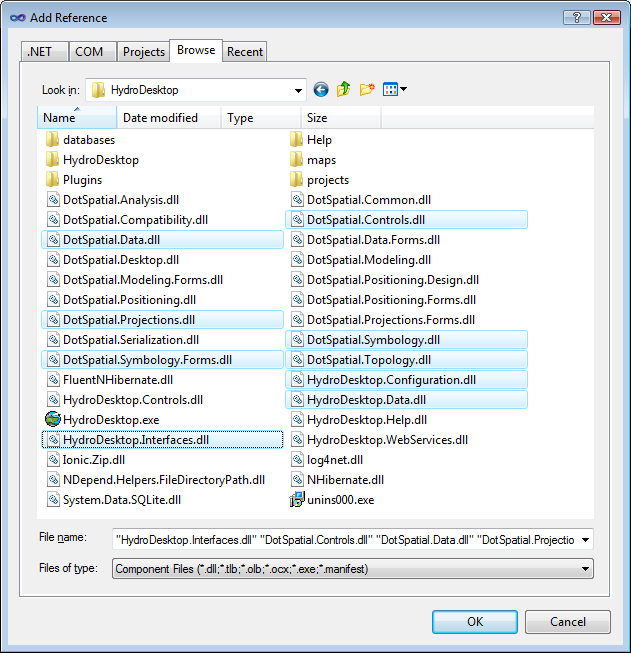


Figure Adding references to DotSpatial and HydroDesktop components

**Important note:** Set the properties of all references: **Copy Local = False** (Fig. 3)

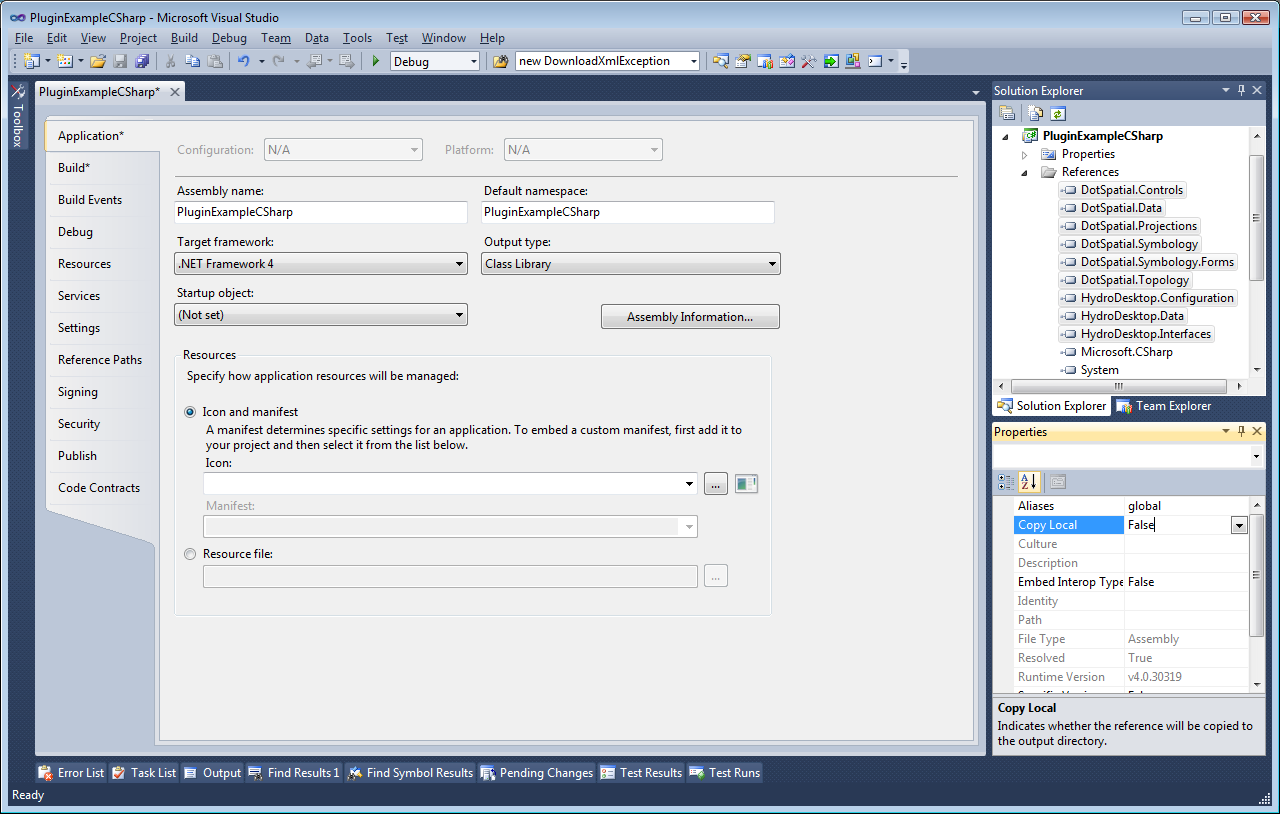


Figure Setting Copy Local = FALSE for all references

Step 3 – Implement the IMapPlugin interface

1. Modify the file AssemblyInfo.cs: In solution explorer click on the > sign in front of **Properties** and double click on AssemblyInfo.cs. At the end of AssemblyInfo.cs, add the following code:

[assembly: DotSpatial.Controls.PluginAssembly]

1. Modify the file Program.cs: add the following using statement to the start of the file:

using DotSpatial.Controls;

using DotSpatial.Controls.RibbonControls;

Delete all code in lines 9 – 14 and replace it by the following code:

public class Main : Extension, IMapPlugin

{

#region IMapPlugin Members

public void Initialize(IMapPluginArgs args)

{

}

#endregion

}

The **Initialize** method will contain code for adding a new buttons to the ribbon toolbar. This code is executed when the user activates the plug-in.

1. Add the **OnDeactivate()** method. This method will contain code for removing any items added by the plugin from the main ribbon toolbar. This code is executed when the user deactivates the plugin.

protected override void OnDeactivate()

{

base.OnDeactivate();

}

1. Add the Plugin attribute just before the declaration of public class Main. This attribute contains the name of the plugin.

//the Main class implements the IMapPlugin interface

[Plugin("Example Plugin")]

public class Main : Extension, IMapPlugin …

1. Press **F6** (**Build** Solution) and start HydroDesktop. If the plugin has been built correctly, you will see a new entry in the HydroDesktop Extensions menu.

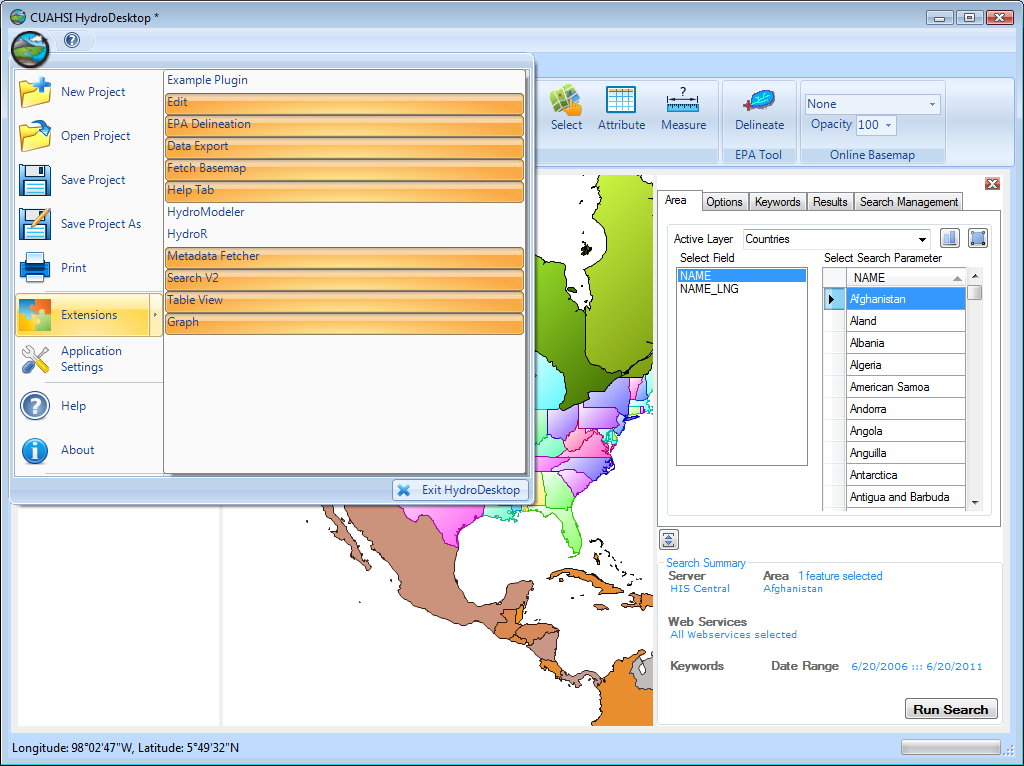


Figure 4 The example plugin is detected by HydroDesktop

Step 4 – Add Ribbon Toolbar Button

This steps shows how to customize the graphical user interface of HydroDesktop. This will add a new button to the main ribbon toolbar.

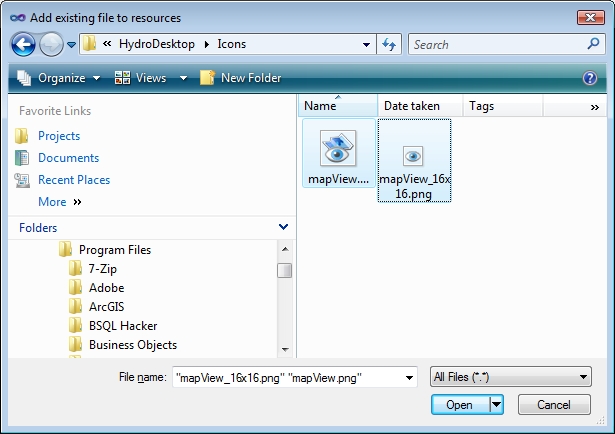
1. Create a default resources file. This file will contain the image displayed in the ribbon toolbar button. In Solution explorer double click Properties. Click on Resources and click the link “This project does not contain a default resources file. Click here to create one.” A new file Resources.resx is created in the Properties folder.
2. Click Add Resource – Add Existing File. Choose the images

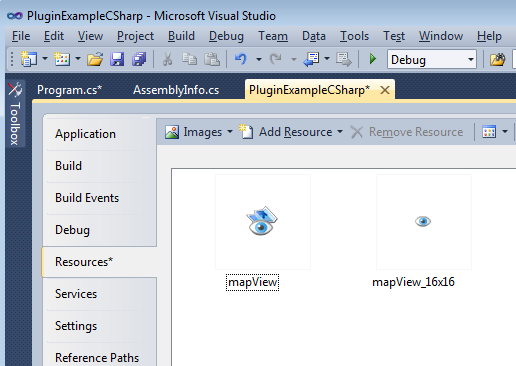
**C:\Program Files\CUAHSI HIS\HydroDesktop\Icons\mapView.png**

and

**C:\Program Files\CUAHSI HIS\HydroDesktop\Icons\mapView\_16x16.png**

**(note:** *you can use any other two png images with sizes 32 x 32 pixels and 16x16 pixels)*





1. Inside the public class Main declare the following class-level global variables:

private RibbonPanel \_myPanel;

private RibbonButton \_myButton;

private IMapPluginArgs \_mapArgs;

These variables are used for future reference for accessing the map and the user interface items.

1. Inside the **Initialize** method write following code:

#region IMapPlugin Members

public void Initialize(IMapPluginArgs args)

{

//the \_mapArgs variable is used for accessing the main app

\_mapArgs = args;

//create a new ribbon panel and add it to the main ribbon

\_myPanel = new RibbonPanel("Example Plugin");

\_myButton = new RibbonButton("QuickMap");

\_myButton.Image = Properties.Resources.mapView;

\_myButton.SmallImage = Properties.Resources.mapView\_16x16;

\_myPanel.Items.Add(\_myButton);

args.Ribbon.Tabs[0].Panels.Add(\_myPanel);

}

#endregion

protected override void OnDeactivate()

{

//remove the ribbon panel and all its items

\_mapArgs.Ribbon.Tabs[0].Panels.Remove(\_myPanel);

base.OnDeactivate();

}

Step 4 – implement the plugin functionality

The “QuickMap” example plugin has the following functionality: Quickly analyze the downloaded data values using the **map**. Why do we need to write this plugin? By default, HydroDesktop only shows the data source organization symbol for each time series location in the map. However, this is only one of many possible uses of the map. An equally important usage of the map is **exploring spatial patterns** in the data values and **geostatistical** analysis. For each location, there is a whole time series (many data values) available. Therefore, it is necessary to **aggregate** the data values by time period at each location. This step is the first step before starting more detailed geostatistical analysis and it is the purpose of the **QuickMap** plugin.

1. **Design a new Windows Form**: In Solution explorer choose Add – New Item and choose Windows Form. Name the form MyForm.cs. Add two DateTimePicker controls, two ComboBox controls, one Button and four Labels to MyForm. Set the following properties of the controls as shown in the table and in the picture:

|  |  |
| --- | --- |
| MyForm.Size | 400, 200 |
| MyForm.Text | Quick Map |
| lblStartDate.Text | Start Date |
| lblEndDate.Text | End Date |
| lblVariableName.Text | Variable Name |
| lblStatistic.Text | Statistic |
| btnCreateMap.Text | Create Map! |

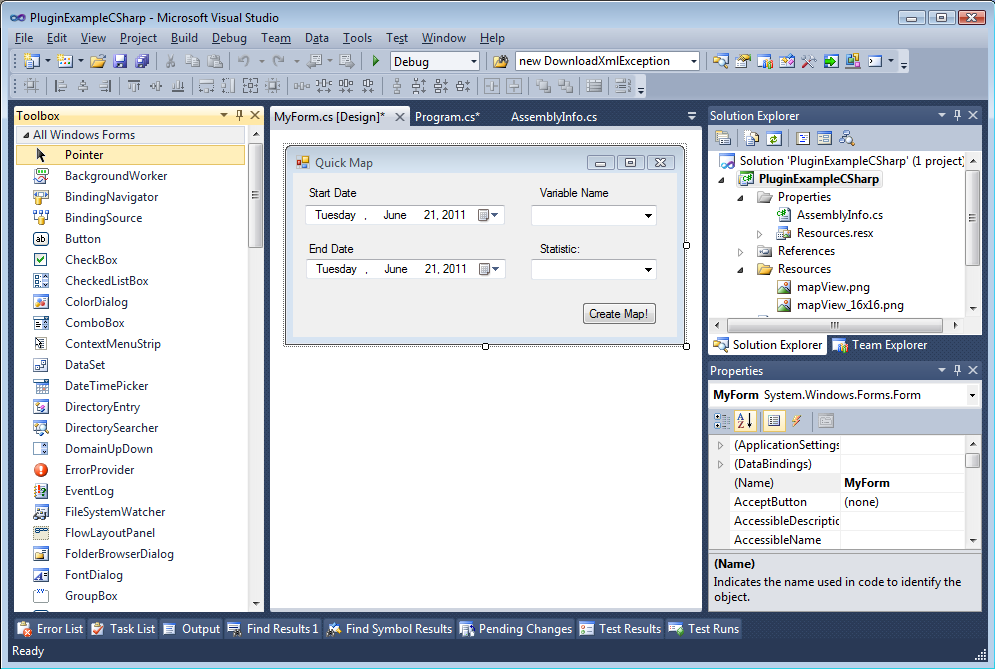


Figure Design View of MyForm

1. Add the following using statements to the start of MyForm.cs:

using DotSpatial.Controls;

using HydroDesktop.Configuration;

using HydroDesktop.Database;

using HydroDesktop.Interfaces;

1. Write the following code in MyForm: Add a MapArgs public property and implement the **MyForm\_Load** Event:

//this property references the main application and main map

public IMapPluginArgs MapArgs { get; set; }

//initializes the combo boxes

private void MyForm\_Load(object sender, EventArgs e)

{

//populate the 'statistic' combo box

cbStatistic.Items.Add("AVG");

cbStatistic.Items.Add("MIN");

cbStatistic.Items.Add("MAX");

cbStatistic.Items.Add("SUM");

cbStatistic.SelectedIndex = 0;

//set default start date to one year ago

dpStartDate.Value = DateTime.Now.Date.AddYears(-1);

//get the available variables

string conString = Settings.Instance.DataRepositoryConnectionString;

DbOperations db = new DbOperations(conString, DatabaseTypes.SQLite);

string sql = "SELECT VariableID, VariableName FROM Variables";

DataTable tab = db.LoadTable(sql);

//data binding

cbVariableName.DisplayMember = "VariableName";

cbVariableName.ValueMember = "VariableID";

cbVariableName.DataSource = tab;

}

1. Implement the ribbon button click event: Modify the source code of the Initialize method in Main.cs class and add a \_**myButton\_click** event handler as follows:

public void Initialize(IMapPluginArgs args)

{

//the \_mapArgs variable is used for accessing the main app

\_mapArgs = args;

//create a new ribbon panel and add it to the main ribbon

\_myPanel = new RibbonPanel("Example Plugin");

\_myButton = new RibbonButton("QuickMap");

\_myButton.Image = Properties.Resources.mapView;

\_myButton.SmallImage = Properties.Resources.mapView\_16x16;

\_myPanel.Items.Add(\_myButton);

args.Ribbon.Tabs[0].Panels.Add(\_myPanel);

//handle the click event

\_myButton.Click += new EventHandler(\_myButton\_Click);

}

//the button click event

void \_myButton\_Click(object sender, EventArgs e)

{

MyForm frm = new MyForm();

frm.MapArgs = \_mapArgs;

frm.Show();

}

Explanation of the source code is: By calling **Settings.Instance.DataRepositoryConnectionString** we obtain the connection string of the current SQLite data repository database. This database contains all time series downloaded by HydroDesktop in the current project. By using the **DbOperations** object we can access the database with an SQL query string and obtain a DataTable of the query result.

1. **Query the Database and create the map layer from query results**

The long SQL query is a join query. It finds rows from the Sites, DataSeries and DataValues table and also performs an aggregate query (AVG, MIN, MAX or SUM). The complete text of the query is (you can copy this into your code):

"SELECT {0}(dv.DataValue) AS '{1}', s.Longitude, s.Latitude, s.SiteName, s.SiteCode, ds.SeriesID FROM DataValues dv, DataSeries ds, Sites s WHERE dv.SeriesID = ds.SeriesID AND ds.SiteID = s.SiteID AND ds.VariableID = {2} GROUP BY ds.SeriesID"

The labels {0}, {1} and {2} are placeholders for the aggregate function (AVG, MIN, MAX or SUM), derived column name and the variable ID.

The complete code listing follows:

private void btnCreateMap\_Click(object sender, EventArgs e)

{

//get the selected statistic and selected variableID

string statistic = cbStatistic.Text;

int variableID = Convert.ToInt32(cbVariableName.SelectedValue);

string valueColumn = String.Format("{0}\_VALUE", statistic);

//form the SQL string

string sqlQuery = String.Format(

"SELECT {0}(dv.DataValue) AS '{1}', s.Longitude, s.Latitude, " +

"s.SiteName, s.SiteCode, ds.SeriesID " +

"FROM DataValues dv, DataSeries ds, Sites s " +

"WHERE dv.SeriesID = ds.SeriesID AND ds.SiteID = s.SiteID AND ds.VariableID = {2} GROUP BY ds.SeriesID",

statistic, valueColumn, variableID);

//connect to the database

string conString =

Settings.Instance.DataRepositoryConnectionString;

DbOperations db = new DbOperations(conString,

DatabaseTypes.SQLite);

DataTable tab = db.LoadTable(sqlQuery);

//export the data table to a shapefile

string variableName = cbVariableName.Text;

string shapeFileName = String.Format(@"{0}\{1}\_{2}.shp",

Settings.Instance.TempDirectory, statistic, variableName);

TableToShapefile(tab, shapeFileName);

//add featureSet to the map

IMapLayer newLayer = MapArgs.Map.Layers.Add(shapeFileName);

//add labels

MapArgs.Map.AddLabels((IFeatureLayer)newLayer,

String.Format("[{0}]", valueColumn), String.Empty,

new LabelSymbolizer(), valueColumn);

}

//This method exports a table to shapefile. The data table must have

//a 'Longitude' and 'Latitude' column.

Private void TableToShapefile(DataTable tab, string shapeFileName)

{

//create a new map layer and add it to the map

FeatureSet fs = new FeatureSet(FeatureType.Point);

fs.DataTable = tab.Clone();

foreach (DataRow dr in tab.Rows)

{

//create a new point and add it to the featureSet

double lon = Convert.ToDouble(dr["Longitude"]);

double lat = Convert.ToDouble(dr["Latitude"]);

DotSpatial.Topology.Point pt =

new DotSpatial.Topology.Point(lon, lat);

IFeature f = fs.AddFeature(pt);

//for each added feature, copy the attribute values

for (int i = 0; i < tab.Columns.Count; i++)

{

f.DataRow[i] = dr[i];

}

}

//set the projection

fs.Projection =

new ProjectionInfo("+proj=longlat +ellps=WGS84 +no\_defs");

fs.Reproject(MapArgs.Map.Projection);

fs.Filename = shapeFileName;

fs.Save();

fs.Dispose();

}

1. **Change the Symbology of the map layer**

The last step demonstrates how to change the size of symbols proportionally to the magnitude of the aggregate of the variable. To add the categories, add following method to myForm.cs:

private void CreateCategories(MapPointLayer layer, string fieldName)

{

//Create a new PointScheme

PointScheme scheme = new PointScheme();

//Set the ClassificationType for the Scheme via EditorSettings

scheme.EditorSettings.ClassificationType = ClassificationType.Quantities;

scheme.EditorSettings.IntervalMethod = IntervalMethod.Quantile;

scheme.EditorSettings.UseSizeRange = true;

scheme.EditorSettings.UseColorRange = true;

scheme.EditorSettings.StartColor = Color.LightBlue;

scheme.EditorSettings.EndColor = Color.DarkBlue;

scheme.EditorSettings.StartSize = 5.0;

scheme.EditorSettings.EndSize = 20.0;

//Set the UniqueValue field name

//Here STATE\_NAME would be the Unique value field

scheme.EditorSettings.FieldName = fieldName;

//create categories based on attributes table and field name

scheme.CreateCategories(layer.DataSet.DataTable);

//Set the scheme to stateLayer's symbology

layer.Symbology = scheme;

}

Finally, call the CreateCategories method from the btnCreateMap\_Click event handler:

private void btnCreateMap\_Click(object sender, EventArgs e)

{

//lines of previous code (in previous listings….)

//add labels

MapArgs.Map.AddLabels((IFeatureLayer)newLayer, String.Format("[{0}]", valueColumn), String.Empty,

new LabelSymbolizer(), valueColumn);

//change map symbology

CreateCategories(newLayer, valueColumn);

}

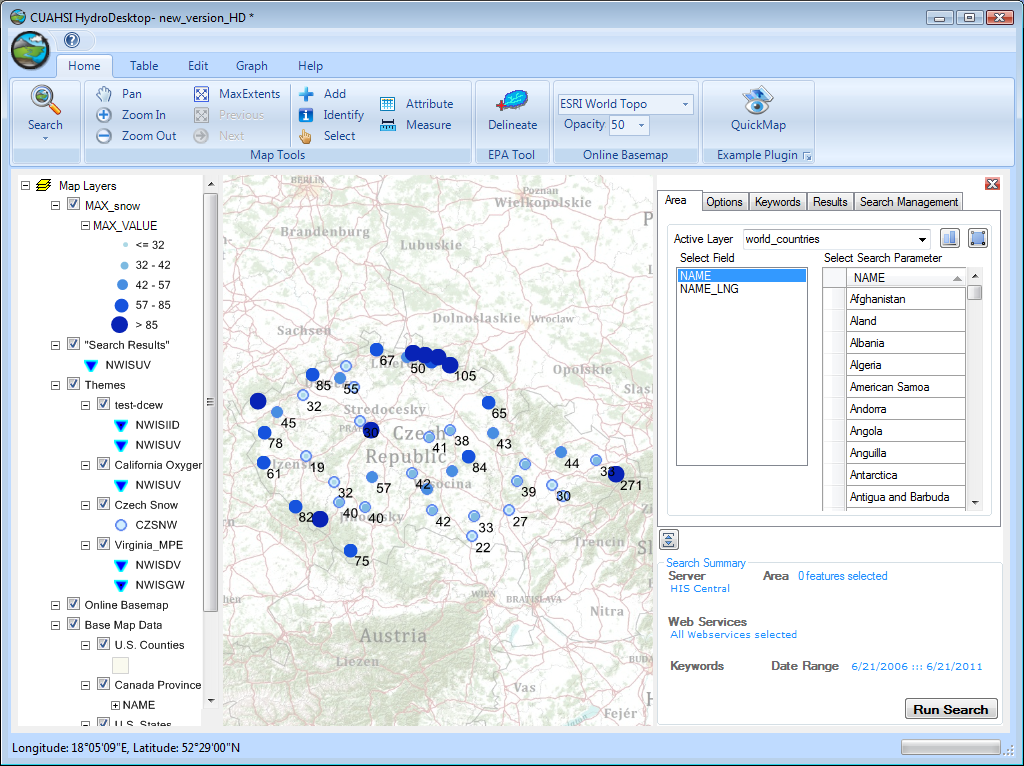


Figure Example of map with labels and categories created by the Quick Map plugin

# Additional programming tasks

The “QuickMap” plugin code is a sample code. It can be expanded for other geospatial analysis functionality. Possibilities of improving the plug-in by using existing HydroDesktop and DotSpatial plugins include:

* Interpolate a continuous grid
* Calculate the average, maximum and minimum of the selected variable in a watershed (for example, average precipitation in each U.S HUC)
* Calculate the surface water balance of a selected watershed
* Use the results of QuickMap to create a thematic map of river streamflow. First, the sites located on each river are found using the buffer and intersection operations. Then the width of each river line is symbolized proportionally to its streamflow. If water quality data is available, then the color of each river line can be symbolized to represent a water quality class.

|  |  |
| --- | --- |
| Initialize(IMapPluginArgs args) | Occurs when the plugin is initialized. The IMapPluginArgs provides access to the map, legend, progress handler and ribbon controls.  Use this method to add new GUI elements belonging to the plugin (Ribbon toolbar buttons, view panels) |
| OnDeactivate() | Occurs when the plugin is deactivated. Use this method to remove any GUI elements belonging to the plugin. |

# Appendix: IMapPluginArgs Properties

The IHydroPluginArgs interface allows the plugin to access the components of the main HydroDesktop application. We recommend storing IHydroPluginArgs as a private class level variable. If other forms or controls of the plugin require access to the main HydroDesktop components, we recommend passing IHydroPluginArgs through the constructor of the Control or Form. Table 3 shows the HydroDesktop application components which are accessible as properties of the IHydroPluginArgs interface:

Table Main Properties of the IHydroPluginArgs interface

|  |  |
| --- | --- |
| Property Name | Property Description |
| Map | The main HydroDesktop map |
| Legend | The map legend of the application |
| Ribbon | The main ribbon control |
| ProgressHandler | The status bar to show progress messages |
| PanelManager | Manager for the main application view area. A plugin can add a new “view” to the main area. |

The properties are explained in detail in the following section.

Map

The Map shows the GIS datasets and the locations of sites with downloaded observation data series. Plug-in can add vector or raster layers to the map, add labels, change the map scale and extent, control the visibility of map layers, select or unselect features, and respond to mouse move and mouse click events on the map. A more detailed documentation of the map component is available on the <http://dotspatial.codeplex.com> website.

Legend

The legend shows the ‘table of contents’ and symbols used for representing geospatial data in the map. It is closely linked with the map. When a layer is added to the map, it automatically appears in the legend. The plugin can respond to mouse click events in the legend and control the legend context menu. A more detailed documentation of the map component is available on the <http://dotspatial.codeplex.com> website.

Ribbon

The ribbon is the main toolbar of the HydroDesktop application. The plugin can extend the ribbon by adding the following items (Figure 1):

* Add a **RibbonTab**
* Add a **RibbonPanel**
* Add a **RibbonButton**
* Add Button to the Quick **Access Toolbar**
* Add a menu item to the **RibbonOrb** menu

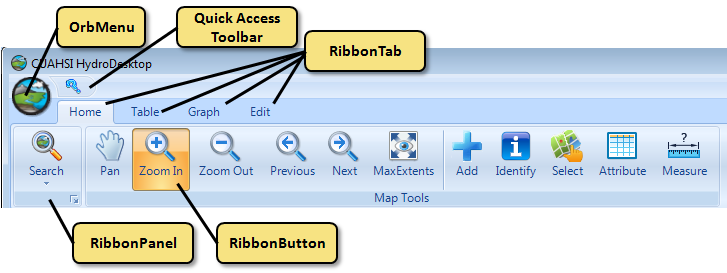


Figure HydroDesktop Ribbon Items

ProgressHandler

The progress handler can be used to report the current status or progress. It is visible in the bottom area of the main application form. The ReportProgress() method is used to show the status message and show the progress percentage in a progress bar.

# Working With HydroDesktop Database

HydroDesktop.Database.DbOperations class

The DbOperations class is used for basic access to the database using SQL Queries to populate the ADO.NET DataTable or DataSet objects. The advantage of the DbOperations class is that it automatically handles the database connection opening and closing. Before executing a SQL query, the connection is opened. When the query is completed, connection is closed immediately. Main properties and methods of the DbOperations class are shown in Table 5.

Table 5 IHydroDbOperations Properties

|  |  |
| --- | --- |
| void ExecuteNonQuery(string sqlString); | Executes a SQL statement without returning any results.  This is used for INSERT or DELETE statements. |
| object ExecuteSingleOutput(String sqlString); | Executes an SQL query with a single output value |
| DataTable LoadTable(string sqlQuery); | Based on a SQL SELECT query, returns a data table with all rows that match the query results |
| DataTable LoadTable(string sqlQuery, DataTable existingTable); | Updates the existing in-memory data table object by the results of the SQL query |
| void SaveTable(string tableName, DataTable table, string primaryKey, string[] uniqueFields); | Inserts the content of the data table back to database. If a row already exists that has the unique fields, then an update is done instead of an insert. The primary key column values are assigned automatically. |
| DbProviderFactory DbFactory { get; } | The database provider factory currently used |
| string ConnectionString { get; } | Get the database connection string |
| bool TestConnection(); | Test if we are able to connect to the database specified by the connection string |
| DbConnection CreateConnection(); | Creates a new instance of a database connection object (use this method when executing multiple commands in a transaction) |
| DbCommand CreateCommand(string txtQuery); | Creates a new instance of a database command (use this in a transaction) |
| DbParameter CreateParameter(DbType parameterType); | Creates a new command parameter with the specified data type (to be used in a transaction) |
| DbParameter CreateParameter(string name, DbType parameterType); | Creates a new instance of a database command parameter with the specified name and data type (to be used in a transaction) |
| DbParameter CreateParameter(DbType parameterType, object value); | Creates a new command parameter with the specified name and value (to be used in a transaction) |
| DbParameter AddParameter(DbCommand cmd, string parameterName, DbType parameterType); | Adds a parameter to an existing command (to be used in a transaction) |

HydroDesktop.Database.RepositoryManagerSQL Class

The RepositoryManager SQL class is used for accessing the database using the HydroDesktop Object Model. For a detailed class diagram of the object model, please use the documentation page: http://hydrodesktop.codeplex.com/wikipage?title=HydroDesktop Object Model

In the current release of HydroDesktop, the main recommended usage of RepositoryManager is **saving data series** to the data repository database and **deleting data series** or themes from the database. The SaveSeries() method has the **OverwriteOptions** parameter (**Copy**, **Append** or **Overwrite**) which controls the handling of existing data values when a series with the same site, variable, method, source and quality control level is found in the database (Table 7).

Table Overwrite options in the SaveSeries() method

|  |  |
| --- | --- |
| OverwriteOptions parameter in SaveSeries() | Description |
| OverwriteOptions.Append | The existing series is modified by appending data values to the end or start of the series. Only data values at times which aren’t present in the existing series are saved to the database |
| OverwriteOptions.Copy | A new series is created in the database. No data values are overwritten. If an existing series with the same properties is found, this method results in two series which only differ by the SeriesID and CreationDateTime. |
| OverwriteOptions.Overwrite | The existing series is modified by deleting all existing data values and replacing them by the new data values. |