

Developing Plugins for HydroDesktop using the IHydroPlugin Interface: Developer Guide

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

August 11, 2010

by:

Jiri Kadlec

Geospatial Software Lab

Idaho State University

Contents

[Introduction 2](#_Toc269391870)

[System Requirements 2](#_Toc269391871)

[Plugin Interfaces for HydroDesktop: 2](#_Toc269391872)

[Getting Started with HydroDesktop Plugins 2](#_Toc269391873)

[IHydroPlugin Interface Attributes and Methods 3](#_Toc269391874)

[IHydroPluginArgs Properties 4](#_Toc269391875)

[Map 5](#_Toc269391876)

[Legend 5](#_Toc269391877)

[Ribbon 5](#_Toc269391878)

[ProgressHandler 6](#_Toc269391879)

[Database 6](#_Toc269391880)

[IHydroPluginArgs.Database.DbOperations Class 6](#_Toc269391881)

[IHydroPluginArgs.Database.RepositoryManager Class 8](#_Toc269391882)

[Views (PanelManager and SeriesView) 8](#_Toc269391883)

[PanelManager 9](#_Toc269391884)

[SeriesView 11](#_Toc269391885)

[SeriesSelector 13](#_Toc269391886)

[Series Selector Properties 13](#_Toc269391887)

[ISeriesSelector.CheckedIDList Property 14](#_Toc269391888)

[ISeriesSelector.VisibleIDList Property 14](#_Toc269391889)

[ISeriesSelector.SelectedSeriesID Property 14](#_Toc269391890)

[ISeriesSelector.FilterExpression Property 14](#_Toc269391891)

[ISeriesSelector.FilterType Property 15](#_Toc269391892)

[ISeriesSelector.ContextMenuStrip Property 15](#_Toc269391893)

[SeriesSelector Events 15](#_Toc269391894)

[ISeriesSelector.SeriesCheck Event 15](#_Toc269391895)

[ISeriesSelector.SeriesSelected Event 16](#_Toc269391896)

[ISeriesSelector.Refreshed Event 16](#_Toc269391897)

[Responding To SeriesSelector Events in The Plugin 16](#_Toc269391898)

# 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. There are two types of plugin which developers can write for HydroDesktop: Map plugin (focused on the map and geographic functionality) and IHydroPlugin (focused on working with the hydrological time series). This guide is a reference manual for the IHydroPlugin interface. It explains the main elements of the interface, shows how to create a HydroDesktop plugins, and demonstrates how to access the HydroDesktop data repository from within a plugin.

# System Requirements

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

* **Microsoft .NET Framework 3.5**
* **Microsoft Visual Studio** (Express version is freely available) or SharpDevelop
* Access to the HydroDesktop source code using **TortoiseSVN**.
* User should be familiar with the Microsoft .NET framework and programming in C# or VB.NET.

# Plugin Interfaces for HydroDesktop:

HydroDesktop supports **2 types of plugins**: IMapPlugin and IHydroPlugin. Table 1 shows how to choose between the IMapPlugin and IHydroPlugin interface when creating a plugin.

Table Plugin interfaces supported by HydroDesktop

|  |  |
| --- | --- |
| Plugin Interface | When to use the interface: |
| IMapPlugin | This is a general type of plugin extending the map and GIS functionality. It can work not only with HydroDesktop, but also in other DotSpatial based application |
| IHydroPlugin | Specialized type of plugin. Only works with HydroDesktop. Provides access to events of the HydroDesktop **database abstraction layer** and the **SeriesSelector component** |

# Getting Started with HydroDesktop Plugins

To get started with a HydroDesktop Plugin, we recommend using one of the **sample plugins** as a template. There are two sample plugins available on the **HydroDesktop SVN repository**:

* RibbonSamplePlugin (c#)
* RibbonSamplePluginVB (VB.NET)

Both plugins show how to add a new ribbon button to the main ribbon toolbar, add a new view panel to the SeriesView, access the time series SeriesSelector menu component and access the HydroDesktop data repository database.

# IHydroPlugin Interface Attributes and Methods

The code of the RibbonSamplePlugin template contains a class named **Main.cs.** The main class inherits from Extension and implements IHydroPlugin. It is has the **Plugin attribute** and two methods: **Initialize** and **OnDeactivate** (code example 1)

[Plugin("SamplePlugin", Author = "ISU", UniqueName = "SamplePlugin\_1", Version = "1")]

public class Main : Extension, IHydroPlugin

{

//private variable to the main HydroDesktop application components

private IHydroPluginArgs \_args;

#region IExtension Members

protected override void OnDeactivate()

{

// Remove all GUI elements added by the plugin

…

// This line is required by the code.

base.OnDeactivate();

}

#endregion

#region IPlugin Members

public void Initialize(IHydroPluginArgs args)

{

\_args = args;

// Add the ribbon buttons, main view and user control

…

}

}

Code Example 1 Structure of the Main class in the Plugin

The Plugin attribute has following properties:

* **Name** (name of the plugins – appears in the HydroDesktop Extensions menu)
* **Author** (author of the plugin)
* **UniqueName**
* **Version**

By using the Plugin attribute, the class library is recognized as a valid plugin by the HydroDesktop application.

The Initialize and OnDeactivate methods are explained in Table2. The **Initialize()** method occurs immediately after the plugin is activated in the Extension menu or when a project file with the plugin name marked as active is loaded. The **OnDeactivate()** method occurs when a plugin is deactivated in the Extensions menu.

Table The Initialize and OnDeactivate Methods

|  |  |
| --- | --- |
| 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. |

# IHydroPluginArgs 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 |
| Database | Provides access to the currently used Data Repository database connection string and database related methods and events |
| PanelManager | Manager for the main application view area. A plugin can add a new “view” to the main area. |
| SeriesView | Manager for the series view area. Plugins which use the SeriesSelector menu should only add their main view to the SeriesView area. |

Please note that IHydroPluginArgs also contains the MainMenu, MainToolStripContainer, Plugins and HydroPlugins property. However, these properties are not being used by HydroDesktop and their value is always set to **null**. 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

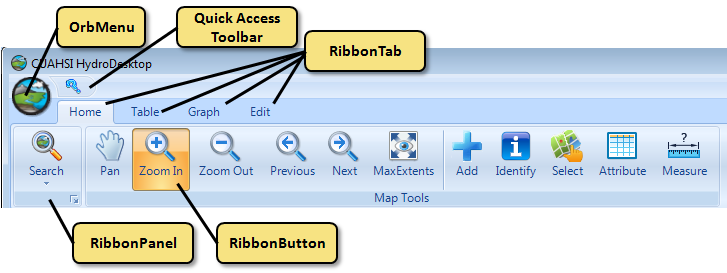


Figure HydroDesktop Ribbon Items

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.

Database

The Database property allows the plugin to directly call methods of the HydroDesktop Database abstraction layer and access the DataRepository database. Three levels of database access are provided:

1. Access using the **Connection string**
2. Access using **SQL queries** (DbOperations). This is recommended for read-only queries.
3. Access using the **HydroDesktop object model** (RepositoryManager). This is recommended for writing values to the database (data import and data editing operations).

The properties and events of IHydroPluginArgs.Database are shown in Table 4.

Table IHydroPluginArgs.Database Properties

|  |  |  |
| --- | --- | --- |
| Property | Type | Description |
| ConnectionString | String | The currently used database connection string |
| DatabaseType | DatabaseTypes | The database type (only SQLite is supported) |
| DbOperations | IHydroDbOperations | The database SQL based query functions |
| RepositoryManager | IRepositoryManager | Methods for saving values from the HydroDesktop object model to the database and object based query functions |
| DatabaseChanged() event |  | Occurs when the database connection string is changed by the user or when a new HydroDesktop project that uses a different database is opened |

IHydroPluginArgs.Database.DbOperations Class

The IHydroDbOperations 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) |

IHydroPluginArgs.Database.RepositoryManager Class

The RepositoryManager 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. |

Views (PanelManager and SeriesView)

The HydroDesktop plugin can modify the main application view area and display its controls in the main view. The plugin may also switch between the views which were added by the main application or by other plugins. Depending on type of plugin, two methods can be used:

* Show control in main view (use the PanelManager class)
* Show control in the series view panel next to the SeriesSelector menu (use the SeriesView class)

Figure 2 explains the relation between PanelManager and SeriesView in the main HydroDesktop application layout.

Figure PanelManager and SeriesView - adding of new view by plugin

PanelManager

The PanelManager contains methods to add the plugin control to the main view and switch between view panels in the main view. The properties and methods of PanelManager are explained in Table 7. The Code Example 2 demonstrates how to add a new view to the main view area. If the plugin uses a separate RibbonTab, then changing the visible panel in the main view should be linked to the **RibbonTab.ActiveChanged** event.

Table Methods, Events and Properties of IHydroPluginArgs.PanelManager

|  |  |
| --- | --- |
| Property, Method of Event | Description |
| string SelectedTabName | The name of the currently visible view |
| int TabCount | The number of view panels |
| void AddTab(string tabName, Control userControl, Image image) | Add a new view to the main view area |
| Void RemoveTab(string tabName) | Remove an existing view from the main view area |
| Void SetControl(string tabName, Control userControl) | Set the user control displayed in the specified view panel of the main view area |
| Event SelectedIndexChanged | Event occurs when the currently visible panel in the main view area is changed |

//reference to the main application and it's UI items

private IHydroPluginArgs \_mapArgs;

//the name of the panel shown in the SeriesView

private string \_panelName = "Sample Plugin";

//the ribbon tab added by the plugin

private RibbonTab \_ribbonTab;

public void Initialize(IHydroPluginArgs args)

{

\_mapArgs = args;

// Add "Ribbon Sample Plugin" panel to the SeriesView

MyUserControl uc = new MyUserControl(\_mapArgs);

\_mapArgs.PanelManager.AddTab(\_panelName, uc);

// Initialize the Ribbon controls in the "Sample Plugin" ribbon tab

\_ribbonTab = new RibbonTab(\_mapArgs.Ribbon, \_panelName);

\_mapArgs.Ribbon.Tabs.Add(\_ribbonTab);

\_ribbonTab.ActiveChanged += new EventHandler(\_ribbonTab\_ActiveChanged);

}

// When the ribbon tab is changed, the Series view panel is changed

void \_ribbonSampleTab\_ActiveChanged(object sender, EventArgs e)

{

if (\_ribbonSampleTab.Active)

{

if (\_mapArgs.PanelManager != null)

{

\_mapArgs.PanelManager.SelectedTabName = \_panelName;

}

}

}

Code Example 2 : Using the IHydroPluginArgs.PanelManager class

The code example 2 is taken from the **main.cs** class of a sample plugin. Three class-level variables are used: \_mapArgs, \_panelName and \_ribbonTab. The \_mapArgs variable allows the plugin to access the Ribbon and PanelManager of the main application. The **MyUserControl** object is a user control which contains the plugin’s user interface elements and occupies the main view. The **ActiveChanged** event occurs every time the user selects a RibbonTab in the main ribbon toolbar. When the user selects the “Sample Plugin” ribbon tab, the view belonging to the sample plugin is shown in the main view.

SeriesView

The SeriesView is a specialized view for plug-ins which need to access the SeriesSelector component. The SeriesSelector component displays the menu with all time **Series** currently available in the HydroDesktop data repository database. Example of plugins using the SeriesView are **TableView**, **GraphView**, **Data Editing View** and **HydroR**. With PanelManager, the plugin user control occupies the whole main view. With SeriesView, the plugin user control only occupies the right hand panel of the main view. The left hand panel is occupied by the SeriesSelector. The SeriesSelector is shared by all plugins in the SeriesView (Figure 3).

Figure Difference between PanelManager and SeriesView

Table Properties, methods and events of the SeriesView

|  |  |
| --- | --- |
| Property, Method of Event | Description |
| string VisiblePanelName { get; set; } | Gets or sets the name of the currently visible plugin panel in the series view. |
| String[] PanelNames {get; } | The names of all plugin panels in the series view |
| ISeriesSelector SeriesSelector { get; } | The SeriesSelector component |
| void AddPanel(string panelName, UserControl control) | Add a new plugin panel to the series view. The plugin panel area is occupied by the user control. |
| Void RemovePanel (string panelName) | Removes an existing plugin panel |
| Event VisiblePanelChanged | Event occurs when the user switches between the panels in the series view |

Code Example 4 demonstrates the usage of SeriesView. The code is taken from the main.cs class in the RibbonSamplePlugin solution. The \_ribbonTab.ActiveChanged event occurs when the user activates the “Ribbon Sample Plugin” ribbon tab. In the event, two steps are requires:

1. Make the SeriesView visible
2. Make the “Ribbon Sample Plugin” panel visible in the series view.

//reference to the main application and it's UI items

private IHydroPluginArgs \_mapArgs;

//the name of the panel shown in the SeriesView

private string \_panelName = "Ribbon Sample Plugin";

//the ribbon tab added by the plugin

private RibbonTab \_ribbonTab;

public void Initialize(IHydroPluginArgs args)

{

\_mapArgs = args;

// Add "Ribbon Sample Plugin" panel to the SeriesView

MyUserControl uc = new MyUserControl(\_mapArgs);

\_mapArgs.SeriesView.AddPanel(\_panelName, uc);

// Initialize the Ribbon controls in the "Ribbon" ribbon tab

\_ribbonTab = new RibbonTab(\_mapArgs.Ribbon, \_panelName);

\_mapArgs.Ribbon.Tabs.Add(\_ribbonTab);

\_ribbonTab.ActiveChanged += new EventHandler(ribbonTab\_ActiveChanged);

}

// When the ribbon tab is changed, the Series view panel is changed

void ribbonTab\_ActiveChanged(object sender, EventArgs e)

{

if (\_ribbonTab.Active)

{

if (\_mapArgs.PanelManager != null)

{

\_mapArgs.PanelManager.SelectedTabName = "SeriesView";

\_mapArgs.SeriesView.VisiblePanelName = \_panelName;

}

}

}

Code Example 3: Using the IHydroPluginArgs.SeriesView class

SeriesSelector

The SeriesSelector component allows the exploration of series available in the HydroDesktop data repository database (Figure 5). The features of the SeriesSelector component are:

* Series selection menu
* Series filter
* Context menu

Figure SeriesSelector control

The SeriesSelector control has properties and events, which allow the plugin to respond to changes in the series checking and selection. The properties, events and methods are explained in detail in the next section. In the code examples, the type of the \_hydroArgs variable is IHydroPluginArgs. The \_hydroArgs variable was populated in the Initialize() method of the plugin (See code example 1).

Series Selector Properties

The SeriesSelector control has the following properties:

* CheckedIDList
* VisibleIDList
* SelectedSeriesID
* FilterType
* FilterExpression
* ContextMenuStrip

### ISeriesSelector.CheckedIDList Property

Get the array of the series IDs whose check boxes are currently checked in the series selection menu. If no series are checked, an empty array is returned. This property is a read-Only property.

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

{

foreach (int seriesID in \_args.SeriesView.SeriesSelector.CheckedIDList)

{

listBox1.Items.Add(seriesID);

}

}

Code Example 4: Display the list of checked series IDs in the listbox

### ISeriesSelector.VisibleIDList Property

Gets the array of the series which are currently accessible in the series selection menu. The visibility of the series depends on the currently used filter expression. If the filter expression is empty string and the filter type is All, then all series from the database are visible.

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

{

foreach (int seriesID in \_args.SeriesView.SeriesSelector.VisibleIDList)

{

listBox1.Items.Add(seriesID);

}

}

Code Example 5: Display the list of checked series IDs in the listbox

### ISeriesSelector.SelectedSeriesID Property

Gets or sets the ID of the series currently selected. Only one series can be selected at one time. The selected series is highlighted by dark blue color in the series selector control. If no series are selected, then the returned value is 0. Use the **SeriesSelected** event to capture changes in the series selection.

private void SetSelectedID()

{

int seriesID = Convert.ToInt32(textBox1.Text);

\_args.SeriesView.SeriesSelector.SelectedSeriesID = seriesID;

}

Code Example 6: Display the list of checked series IDs in the listbox

### ISeriesSelector.FilterExpression Property

Gets the filter expression currently used to restrict the number of series check boxes in the series selection menu. The currently used filter expression can be seen by checking the “Complex Filter” filter option in the series selector menu. If the filter expression is empty string, then all series are shown. Multiple properties can be used in the filter expression. The filter expression is specified by the user using the simple filter of complex filter.

### ISeriesSelector.FilterType Property

Get the filter type of the currently used filter. It can be All (all series are shown), Simple (filter based on Theme, Site, Variable, Method, Source or QualityControlLevel) or Complex (filter is based on a combination of multiple series properties). The filter type is automatically determined from the filter expression.

### ISeriesSelector.ContextMenuStrip Property

The context menu is displayed when right-clicking on the series in the series selection menu. The plug-in can add a new context menu item to the context menu and respond to the ContextMenuItem.Click event.

SeriesSelector Events

The SeriesSelector control has the following events:

* SeriesCheck
* SeriesSelected
* Refreshed

### ISeriesSelector.SeriesCheck Event

This event fires when a series check box is checked or unchecked by the user. If the user clicks the ‘Uncheck All’ button and some series were checked, then the SeriesCheck event occurs for each of the series being unchecked. To pass the checked state and the series ID, the event arguments are used:

public SeriesEventArgs(int seriesID, CheckState checkState)

The **seriesID** is the ID of the currently checked or unchecked series. The **checkState** can have the value Checked (series has been checked) or Unchecked (series has been unchecked).

void SeriesSelector\_SeriesCheck(object sender, SeriesEventArgs e)

{

int seriesID = e.SeriesID;

CheckState checkState = e.CheckState;

if (checkState == CheckState.Checked)

{

MessageBox.Show("Checked Series: " + seriesID);

}

else

{

MessageBox.Show("Unchecked Series: " + seriesID);

}

}

Code Example 7: SeriesCheck Event

### ISeriesSelector.SeriesSelected Event

This event occurs when the user left – clicks on the series row in the series selection menu. Only one series may be selected at one time. Use the SelectedSeriesID property to get the ID of the currently selected series.

void SeriesSelector\_SeriesSelected(object sender, EventArgs e)

{

int seriesID = \_args.SeriesView.SeriesSelector.SelectedSeriesID;

MessageBox.Show("Selected Series: " + seriesID);

}

Code Example 8: SeriesSelected Event

### ISeriesSelector.Refreshed Event

This event occurs when the whole series selection menu has been forced to refresh as a result of changes in the database (a theme has been added or deleted, some series have been added or deleted, the database connection has changed). When this event occurs, the plugin should re-calculate all its properties which rely on the SeriesSelector.CheckedIDList and SeriesSelector.SelectedSeriesID properties.

Responding To SeriesSelector Events in The Plugin

The plugin can choose from three strategies how to respond to the SeriesCheck, SeriesSelected and Refreshed events of the SeriesSelector. The choice depends on performance considerations. Table 10 shows the overview of options.

|  |  |
| --- | --- |
| Option | Considerations |
| Handle the **SeriesSelector.SeriesCheck** event | Immediately respond to the user checking or unchecking a series, even if it was checked or unchecked from a different plug-in |
| Handle the **SeriesSelector.SeriesCheck** event only when the current plugin panel is **visible** | Only respond to the user checking or unchecking a series, if the series was checked or unchecked from the current plug-in. In this case, it is also necessary to use the panelVisibleChanged event to synchronize the plugin with other plugins when its panel becomes visible. |
| Handle the **SeriesView.VisiblePanelChanged** event | Only refresh controls of the plugin based on changes in the SeriesSelector menu when the user switches between visible panels in the SeriesView. |
| Handle a custom **Button.Click** event | If the operations of the plugin are time consuming, the plugin can create a ‘Refresh’ of ‘Calculate’ button and only perform the operation based on the SeriesSelector.CheckedIDList when the button is clicked by the user. |