ArcLogistics Tutorials: How to create an Extension plugin

This document explains how to create an *Extension* type of plugin for ArcLogistics. Before starting plugin development, please read the document titled **ArcLogistics Plugins: An Introduction**, to have a firm understanding of the basic concepts, and the document titled **ArcLogistics Plugins: Getting Started**, for a step-by-step guide for creating a simple "Hello World" plugin.

Setting up the project environment

1. Start Visual Studio, and create a new C# project of *Class Library* type. Change the target framework to **.Net Framework 4.0** if it is not already selected.

2. To make debugging easier, change the *Build* settings by providing the **ArcLogistics install path** as the *Output path* for the plugin. This will cause the plugin dll to be built in the install directory, where it is automatically activated and is ready to be used.

3. Also, change the *Start Action* setting under the *Debug* tab to **Start External Program** and select the **ESRI.ArcLogistics.App.exe** file from the install directory. This will launch ArcLogistics with the plugin enabled when you start debugging in Visual Studio.

Implementation

1. Add references to the following two ArcLogistics files from the install directory: **ESRI.ArcLogistics.App.exe** and **ESRI.ArcLogisticsNG.dll**. You may need to add more references as your development progresses.

2. Add the following lines of code to the top of the source file:
```
using System;
using ESRI.ArcLogistics.App;
using ESRI.ArcLogistics.Routing;
```
You may need to add more "usings" as your development progresses.

3. Add the `IExtension` Interface as a superclass for the plugin class, as shown below:
```
public class MyCustomExtension : IExtension
```

4. Right-click `IExtension` and choose **Implement Interface**. This adds stubs for one method and two properties.

5. Modify the stubs in a manner similar to the code sample given below. Also add the two event handlers. The *Description* property specifies the text which is stored as the plugin's description. The *Initialize* method contains code for subscribing to the *ApplicationInitialized* event. The *Name* property is used to provide the name of the plugin. The remaining two methods are the event handlers. The *Current_ApplicationInitialized* event handler waits for the ArcLogistics application to finish loading, and then subscribes to the *AsyncSolveCompleted* event. The Solver_*AsyncSolveCompleted* event handler displays a message in the application's message window each time a Solve operation is completed. For a list of some other useful events, look at the list at the end of this document.

```
public class MyCustomExtension : IExtension
    {
        public string Description
        {
            get { return "This extensions shows an alert when a Build Routes
                          operation is completed."; }
        }

        public void Initialize(App app)
        {
            App.Current.ApplicationInitialized += new
                                    EventHandler(Current_ApplicationInitialized);
        }

        public string Name
        {
            get { return "ExtensionPluginTutorial.MyCustomExtension"; }
        }

        private void Current_ApplicationInitialized(object sender, EventArgs e)
        {
            App.Current.Solver.AsyncSolveCompleted += new
                     AsyncSolveCompletedEventHandler(Solver_AsyncSolveCompleted);
        }

        private void Solver_AsyncSolveCompleted(object sender,
                                                  syncSolveCompletedEventArgs e)
        {
            App.Current.Messenger.AddInfo("Solve completed");
        }
    }
```

Testing

1. Build the solution. If build succeeds, the plugin dll should appear in the application install directory.

2. Either run ArcLogistics, or start debugging the solution from Visual Studio. In either case, the application will start, load the plugin and activate it automatically.

3. The event handlers will listen for the subscribed events, and perform their tasks when the events occur.

Wrapping up

Congratulations! You have just learned how to create *Extension* plugins for ArcLogistics. To learn how to create other kinds of plugins and adding more functionality to ArcLogistics, go through the other tutorials and various sample projects available on the *ArcLogistics Resource Center*. Refer back to the document titled **ArcLogistics Plugins: An Introduction** for a complete list and descriptions of all resources available for you to create your own plugins.

Appendix A: List of useful events

Event	Description
App.*ProjectLoaded*	Raised when a project is opened.
App.*ProjectClosing*	Raised before a project is closed.
App.*ProjectClosed*	Raised when a project is closed.
App.*ApplicationInitialized*	Raised after application initializes itself during startup.
App.*CurrentDateChanged*	Raised when user changes current date on a calendar.
App.Project.*SavingChanges*	Raised when Project.Save() method is called, but before changes are saved to the project database.
App.Project.*SaveChangesCompleted*	Raised when changes are saved to the project database.
App.Solver. *AsyncSolveStarting*	Raised when the user starts a routing operation, before the actual processing starts.
App.Solver. *AsyncSolveStarted*	Raised when processing starts for a routing operation.
App.Solver. *AsyncSolveCompleted*	Raised when a routing operation is completed.
CollectionChanged	Raised by some classes (such as *DataObject*) when their member collections are changed.
PropertyChanged	Raised by some classes (such as *DataObject*) when their properties are changed.