ArcLogistics Tutorials: How to create a Command plugin

This document explains how to create a *Command* 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.Windows.Forms;
   using ESRI.ArcLogistics.App.Commands;
   ```
   You may need to add more "usings" as your development progresses.

3. Add the following line of code just above the class declaration for the Command plugin:
   ```
   [CommandPlugIn(new string[1] { "Location1" })]
   ```
   This registers the new command with ArcLogistics, so that the button for this new command will appear in the appropriate place specified by the "Location1" parameter. Look at the table at the end of this document for a list of allowed values, and the locations they represent in the software. To make a command appear in two or more places, use the following (or similar) line of code:
   ```
   [CommandPlugIn(new string[2] { "Location1" , "Location2" })]
   ```
   For example, replacing `"Location1"` with `"ScheduleTaskWidgetCommands"` and `"Location2"` with `"UnassignedOrdersRoutingCommands"` will cause the plugin button to appear in the *Tasks* widget of the *Schedule* tab's *Optimize and Edit* action panel, and also on the button group of the *Orders* view.

4. Add the `ICommand` Interface as a superclass for the command class, as shown in the following example:
   ```
   public class CommandPluginCmd : ESRI.ArcLogistics.App.Commands.ICommand
   ```

5. Right-click *ICommand* and choose **Implement Interface**. This adds stubs for two functions and five properties.

6. Modify the stubs in a manner similar to the code sample given below. The *Execute* method contains the code which must be run when the button for the Command is clicked. The *Initialize* method contains code for initializing class members. The *IsEnabled* property determines if the button is enabled in the UI. The *KeyGesture* property specifies the key combination which can trigger the command. The *Name* property is used to provide the name of the plugin. The *Title* Property specifies the text that appears on the command's button(s). The *TooltipText* property determines the tooltip which is displayed when the user's mouse hovers over the command's button(s).

```
 public class CommandPluginCmd : ESRI.ArcLogistics.App.Commands.ICommand
{
    #region ICommand Members

    public void Execute(params object[] args)
    {
        MessageBox.Show("Hello, World!");
    }

    public void Initialize(ESRI.ArcLogistics.App.App app)
    {
    }

    public bool IsEnabled
    {
        get { return true; }
    }

    public System.Windows.Input.KeyGesture KeyGesture
    {
        get { return null; }
    }

    public string Name
    {
         get { return "CommandPluginTutorial.CommandPluginCmd"; }
    }

    public string Title
    {
        get { return "Say Hello"; }
    }

    public string TooltipText
    {
        get { return "Hello World Tooltip"; }
    }

    #endregion
}
```

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. Once the application finishes starting up, one (or more) new button(s) will appear at the location(s) you specified.

4. Clicking the button(s) will run the code in the execute method and perform the specified action.

Wrapping up

Congratulations! You have just learned how to create *Command* 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: Allowed values for specifying Command locations

Location	String to use
Tasks widget of License action panel on Home tab	“LicenseTaskWidgetCommands”
Tasks widget of Optimize and Edit action panel on Schedule tab	“ScheduleTaskWidgetCommands”
Button Group on Orders view of Optimize and Edit action panel on Schedule tab	“UnassignedOrdersRoutingCommands”
Button Group for Stops on Routes view of Optimize and Edit action panel on Schedule tab	“RouteRoutingCommands”
Button Group for Routes on Routes view of Optimize and Edit action panel on Schedule tab	“RoutesRoutingCommands”
Button Group on Time view of Optimize and Edit action panel on Schedule tab	“TimeViewRoutesRoutingCommands”
Button group of Fuel Types action panel on Setup tab	“FuelTypesCommands”
Button group of Vehicles action panel on Setup tab	“VehiclesCommands”
Button group of Drivers action panel on Setup tab	“DriversCommands”
Vehicle Specialties Button group of Specialties action panel on Setup tab	“VehicleSpecialtiesCommands”
Driver Specialties Button group of Specialties action panel on Setup tab	“DriverSpecialtiesCommands”
Button group of Locations action panel on Setup tab	“LocationsCommands”
Button group of Mobile Devices action panel on Setup tab	“MobileDevicesCommands”
Button group of Zones action panel on Setup tab	“ZonesCommands”
Button group of Barriers action panel on Setup tab	“BarriersCommands”
Button group of Default Routes action panel on Setup tab	“DefaultRoutesCommands”
Tasks widget of Projects action panel on Home tab	“ProjectTaskWidgetCommands”