Using Albo1125's APIs

A quick guide to using Albo1125's APIs in your own LSPDFR plugins.
By Albo1125. Last revised 22/05/2016.

Requirements & Credits

- This guide assumes your plugin is an LSPDFR plugin (this means that it's installed in the Plugins/LSPDFR folder). It is possible to access the my APIs from a standalone RPH plugin, however this is extremely complicated and beyond the scope of this guide.
- This will only work using LSPDFR 0.3.1 or above.
- General reminder: you must not bundle any of my APIs or plugins with any plugins you release. This guide will help you make sure your plugin won't crash if the end user doesn't have my plugin installed.
- Thank you LMS for the amazing help to make this possible.

Getting started & Preparing…

In this guide, I will show you how to access the Traffic Policer API as an example. However, the concepts I demonstrate can be easily used to access all my other APIs too.

Step 1

Add Traffic Policer.dll as a reference in your project. This can be done by clicking the 'Properties' menu and selecting 'Add a Reference' in Visual Studio. In the window that appears, select 'Browse' and select Traffic Policer.dll.

Step 2

Add the following event handler to your project. I recommend putting it in the same class as where your other initialisation code is located. You'll need to add **using System.Reflection;** and **using LSPD_First_Response.Mod.API;** at the top of your .cs file too.

```
public static Assembly LSPDFRResolveEventHandler(object sender, ResolveEventArgs args)
{
    foreach (Assembly assembly in Functions.GetAllUserPlugins())
    {
        if (args.Name.ToLower().Contains(assembly.GetName().Name.ToLower()))
        {
            return assembly;
        }
    }
    return null;
}
```

When this event handler is called, it will loop through all the plugins loaded by LSPDFR itself (i.e. plugins from the Plugins/LSPDFR folder). When it finds the requested plugin (by name) it returns it. If the requested plugin is not found, it returns null (and your plugin will most likely crash).

Step 3

Add the following line of code to your existing initialisation code:

```
AppDomain.CurrentDomain.AssemblyResolve += new ResolveEventHandler(LSPDFRResolveEventHandler);
```

This subscribes the event handler we added before to the AssemblyResolve event. This event only occurs if an external assembly (.dll reference) cannot be found.

Step 4

Add the following method to your code in an easily accessible class:

```
public static bool IsLSPDFRPluginRunning(string Plugin, Version minversion = null)
{
    foreach (Assembly assembly in Functions.GetAllUserPlugins())
    {
        AssemblyName an = assembly.GetName();
        if (an.Name.ToLower() == Plugin.ToLower())
        {
             if (minversion == null || an.Version.CompareTo(minversion) >= 0)
             {
                  return true;
             }

         }
     }
     return false;
}
```

This method checks if an assembly is installed by their name and should be called beforehand to check if the user is running the desired plugin (e.g. Traffic Policer). The name of a plugin is typically its file name without the .dll extension. Optionally, you can also pass a Version object that ensures the user is running the version required for the Traffic Policer API to work properly.

Note: If this is called right when your plugin loads, LSPDFR might not have loaded all its plugins yet. Thus, it might return false if the plugin you're requesting hasn't been loaded by LSPDFR yet. I recommend waiting a few (milli)seconds after LSPDFR initialisation before calling this in your code to ensure this issue doesn't occur.

Accessing the Traffic Policer API safely

Step 5

Create an entirely new static class. Name it something like TrafficPolicerAPIFunctions. This class will be used to access the Traffic Policer API safely. All references to Traffic Policer should be done in this 'wrapper class' to prevent people without Traffic Policer installed experiencing issues.

***Note**:* Alternatively, you can also create a new class with a new, separate implementation of your plugin's functionality along Traffic Policer's functions included there.

If this is only used if you're sure Traffic Policer exists beforehand, you do not need to create another 'wrapper class'. For the sake of this guide, though, I will show you how to implement Traffic Policer's API functions without editing your original code too much. If you know what you're doing you can deviate from this guide here.

Step 6

An example of the wrapper class is as follows. As you can see, Traffic Policer API methods are only used in this class, and not in the rest of your code. In your other code, you will only call methods of the wrapper class you're creating to prevent the user from crashing if they don't have Traffic Policer installed.

```
    using Traffic_Policer;
    using Traffic_Policer.API;
    using Rage;
    static class TrafficPolicerFunctions
    {
        public static bool IsVehicleInsured (Vehicle veh)
        {
            return Traffic_Policer.API.Functions.IsVehicleInsured(veh);
        }
        public static void DisplayInsuranceStatusForVehicle(Vehicle veh)
        {
            bool insurancestatus = IsVehicleInsured(veh);
            switch (insurancestatus)
            {
                case true:
                    Game.DisplayNotification("~b~Mors Mutual Insurance Status:
~g~INSURED");
                    break;
                case false:
                    Game.DisplayNotification("~b~Mors Mutual Insurance Status:
~r~UNINSURED");
                    break;

            }
        }
        public static void SetPedDrugsLevels(Ped ped, bool Cannabis, bool Cocaine)
        {

            Traffic_Policer.API.Functions.SetPedDrugsLevels(ped, Cannabis, Cocaine);
        }
    }
```

"Why do I need a wrapper class?" you may ask. If the compiler detects a Traffic Policer API method that's within a class where a method is currently being executed, it checks to see if Traffic Policer is installed. Should a user not have it installed, an exception will be thrown resulting in an error. To allow users without Traffic Policer to still use your plugin, you put all references to the actual Traffic Policer API in your wrapper class. That way, the compiler only checks if Traffic Policer is installed when you call the wrapper class from your other code (and if you do that, you should be sure Traffic Policer is installed beforehand as shown below).

Step 7

As an example, I will now show you how to change a ped's drug levels. Important to note here is that this is only done if Traffic Policer is actually running and if the user is running Traffic Policer 6.8.0.0 or higher:

```
Ped TestPed = new Ped(Game.LocalPlayer.Character.GetOffsetPositionFront(4f));
TestPed.BlockPermanentEvents = true;
TestPed.IsPersistent = true;

if (IsLSPDFRPluginRunning("Traffic Policer", new Version("6.8.0.0")))
{
      TrafficPolicerFunctions.SetPedDrugsLevels(TestPed, true, true); //Wrapper Method
}
```

The importance of having Traffic Policer's API functions in a separate class is demonstrated too. Say an end user doesn't have Traffic Policer installed. If you had directly called the Traffic Policer API function in your if statement, your plugin would have crashed at runtime because the runtime handler would have detected Traffic Policer was required. By putting all calls to the Traffic Policer API in a separate class, you're basically hiding that code from the runtime handler until you're sure the end user is running Traffic Policer.

Wrapping up

That's really all there is to it! I will keep improving my APIs over time. If you have any questions please feel free to message me.