# Practice Velocity – Coding Problem

Scott Rupke

5/18/2015 – Initial Document

## Tools Used

Visual Studio 2013

.NET Framework 4.5

## Overview

The document explains what has been done to solve the coding problem.

## Solution Explorer Diagram

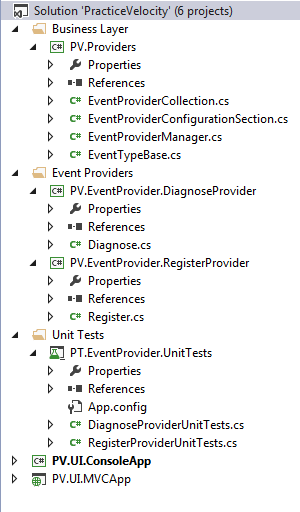
Above is an expanded look into the visual studio solution. Notice that the solution is organized in folders.

The “Business Layer” folder holds all the projects that are referenced in all the other projects.

The “Event Providers” folder holds all the business logic for each provider that was created to solve the coding problem.

The “Unit Tests” folder holds all the unit tests for the projects within the solution.

The main applications of the solution are the PV.UI.ConsoleApp and PV.UI.MVCApp to demonstrate the use of the Event providers.



## PV.Providers project

This project was the first set in the process to solve the code problem. The framework is defined here to dynamically allow a developer to add a provider solution to an existing application.

### EventProviderConfigurationSection.cs

This class is required for the app.config file to work correctly. It defines how the section in the app.config is constructed.

### EventProviderCollecton.cs

This class is used to get a handle to the list of providers in the app.config file.

### EventProviderManager.cs

This class is a static class that is used in the software to access the provider functionality. This is demonstrated in the PV.UI.ConsoleApp program.cs

### EventTypeBase.cs

This is the base abstract class that needs to be inherited from in order to create a new Event Provider.

## PV.EventProvider.DiagnoseProvider project

This project solves the code problem for the “Event Type = Diagnose” rule. The business logic is contained in the GetEventString method within the Diagnose class. Notice that there is a post build script to copy the dll to the Debug directory for console app to work correctly. The console app also requires the following setting in the app.config.

<configuration>

  <configSections>

    <section name ="EventProvider" type="PV.Providers.EventProviderConfigurationSection, PV.Providers"/>

  </configSections>

  <EventProvider>

    <providers>

      <add name="Diagnose" description="Diagnose Provider" type="PV.EventProvider.DiagnoseProvider.Diagnose, PV.EventProvider.DiagnoseProvider"/>

      <add name="Register" description="Register Provider" type="PV.EventProvider.RegisterProvider.Register, PV.EventProvider.RegisterProvider"/>

    </providers>

  </EventProvider>

    <startup>

        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

    </startup>

</configuration>

## PV.EventProvider.RegisterProvider project

This project solves the code problem for the “Event Type = Register” rule. The business logic is contained in the GetEventString method within the Register class. Notice that there is a post build script to copy the dll to the Debug directory for console app to work correctly. The console app also requires the following setting in the app.config.

<configuration>

  <configSections>

    <section name ="EventProvider" type="PV.Providers.EventProviderConfigurationSection, PV.Providers"/>

  </configSections>

  <EventProvider>

    <providers>

      <add name="Diagnose" description="Diagnose Provider" type="PV.EventProvider.DiagnoseProvider.Diagnose, PV.EventProvider.DiagnoseProvider"/>

      <add name="Register" description="Register Provider" type="PV.EventProvider.RegisterProvider.Register, PV.EventProvider.RegisterProvider"/>

    </providers>

  </EventProvider>

    <startup>

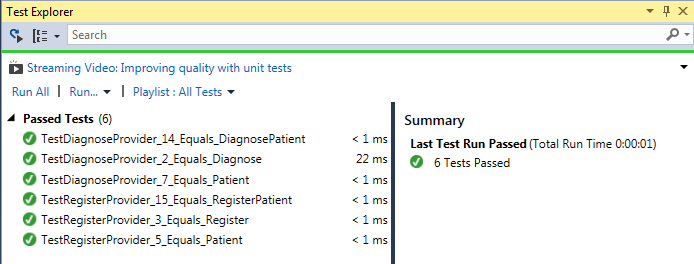
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />

    </startup>

</configuration>

## PT.EventProvider.UnitTest project

This project was used to verify the rules that were presented. Below is a screenshot of the results of the Unit Tests.



## Steps to create a new “Event” provider.

1. Right click on the solution and add a new “Class Library” project.
2. Give the project the naming convention “PV.EventProvider.???Provider” where ??? is the name of the “Event Type”.
3. Rename the Class1.cs file to equal the ??? value and modify the code to look like the below.

public class Class1 : EventTypeBase

{

    /// <summary>

    /// This method returns the appropriate string value based upon the business logic.

    /// </summary>

    /// <param name="value">The current value.</param>

    /// <returns>string</returns>

    public override string GetResultString(int value)

    {

        // Insert the business logic here.

        return base.GetResultString(value);

    }

}

1. Add the PV.Providers project as a reference.
2. Add the following Post Build script to the project in order for it to be used in the Console application.

Copy "$(TargetPath)" "$(SolutionDir)PV.UI.ConsoleApp\bin\Debug\$(TargetFileName)"

Copy "$(TargetPath)" "$(SolutionDir)\PV.UI.MVCApp\bin\$(TargetFileName)"

1. Build the project.
2. Modify the app.config file to add the new provider to the Console App project.
3. Modify the web.config file to add the new provider to the MVC App project.
4. Add the project to the unit test project as a reference.
5. Modify the app.config file to add the new provider to the Unit Test project.