Practical C# Coding Standards

File Organization

John F. Gnazzo

Introduction

A software coding standard is a set of rules and guidelines for the formatting and organizing source code. Software coding standards are used to define a specific programming style.

In professional environments, the benefits of coding standards include readability, maintainability and compatibility. Any member of a development team needs to be able to quickly read and understand the code of another member. The developer who maintains a piece of code tomorrow may not be the developer who programmed it today. Many of today's enterprise solutions are so complex that multiple development teams commonly unite to build a singular enterprise software application. With coding standards in place, disparate teams can rely on the way that they can interface with the code built by a separate team.

This blog will touch on a simple set of practical coding standards for the **File Organization** of a software program coded in C#, a fourth generation language commonly used by developers to develop desktop, mobile, and web applications.

File Organization

C# code should be written in files with the .cs suffix, and have the following format illustrated in the following sections

Using statements

Using statements are used to define the assemblies that will be used by the file.

```
using System;
using System.Linq;
using MyVision.Services.Data;
```

Name Space Definition

A Name Space is used to eliminate the possibility of a class collision, i.e. two code bases having a liked named class.

Name Space Definition

```
namespace MyVision.Business
```

Class Definition

Used to describe the name of the class and optionally define the base class inherited from and/or interfaces implemented

Base Class

```
class SettingsController {
```

Class inheriting from Form Class

```
public partial class BiddingAdminForm : Form
```

Class inheriting from Form Class and implementing IDoit Interface

```
public partial class BiddingAdminForm : Form, IDoit
```

Constants

Preface with private access modifier and readonly attribute.

Capitalize all letters in constant.

```
private readonly decimal PI = 3.14159;
```

Class Variables

Preface with private access level and add _ prefix to variable name. variable name should not be capitalized.

```
private MyVisionEntities _context;
```

Properties

Preface with public, protected or Internal access modifiers. Capitalize property name.

```
public Settings Settings { get; set; }
```

Constructor

Preface with public access modifier. Name must equal name of class. May optionally contain one or more parameters.

Empty Constructor

```
public BiddingAdminForm()
```

Constructor containing parameter

```
public SettingsController(MyVisionEntities context, Settings settings)
```

Functions

Preface with appropriate access modifier and return type. Always decorate with function comment

```
/// <summary>
/// Gets Default Labor Rate from Settings Table
/// </summary>
/// <returns></returns>
public decimal GetDefaultLaborRate()
{
        /// <summary>
        /// Gets Default Labor Rate from Settings Table
        /// </summary>
        /// <returns></returns>
```

```
        public decimal GetDefaultLaborRate()
        {
            try
            {
                var settings = _context.Settings.FirstOrDefault();
                if (settings != null)
                    return settings.DefaultLaborRatePerHour;
            }
            catch (Exception)
            {
                return StandardLaborRate;
            }
            return StandardLaborRate;
        }
```

Events

Preface with appropriate access modifier usually private and void return type. Always decorate with function comment

```
/// <summary>
/// Executes when Search Form OK Button Clicked
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void buttonSearchFilterFormOK_Click(object sender, EventArgs e)
{
     var result = MessageBox.Show(@"Are you Sure you wish to proceed?",
          @"Create a new Job", MessageBoxButtons.OKCancel,
          MessageBoxIcon.Question);

          DialogResult = result;

     if (result == DialogResult.Cancel)
     {
          MessageBox.Show(@"Convert to Job Canceled", @"Create a new Job",
          MessageBoxButtons.OK, MessageBoxIcon.Information);
     }
}
```

Example C# File

```
using System;
using System.Linq;
using MyVision.Services.Data;

namespace MyVision.Business
{
    class SettingsController
    {
        private readonly decimal PI = 3.14159;
        private MyVisionEntities _context;
        public Settings Settings { get; set; }

        public SettingsController(MyVisionEntities context, Settings settings)
     {
           _context = context;
           Settings = settings;
        }

        /// <summary>
        /// Gets Default Labor Rate from Settings Table
        /// </summary>
        /// <returns></returns>
        public decimal GetDefaultLaborRate()
        {
            try
            {
                var settings = _context.Settings.FirstOrDefault();
                if (settings != null)
                    return settings.DefaultLaborRatePerHour;
            }
            catch (Exception)
            {
                return StandardLaborRate;
            }
            return StandardLaborRate;
        }

    }
}
```

Conclusion

The benefits of Coding Standards include readability, maintainability and compatibility. This blog illustrated a practical coding standard for file organization.