The C# Attribute

Source	http://www.csharphelp.com/archives3/archive558.html

There are numerous ways designers have coupled information with classes, fields, members of a class. Category of a property, transaction context for a method, persistence mapping, keywords to extend programming languages, using external files, etcetera, have been suggested by Anders Hejlsberg, Microsoft. Each of these have advantages over each other and along with them, various drawbacks that need to be hurdled to come up with a usable solution.

While working for a previous employer, a major Dutch company that no longer exists, we created a programming environment for application programmers, where we wanted to hide the database details (including but not limited to which database was used) behind object-oriented programming. The idea was simple: Application programmers (people using our environment) could create Java applications quickly without having to worry about database specific details. They would also maintain metadata about classes in a XML-file per Java class, using a user interface. We developed an XML based development repository to store this metadata (XML files). The programming environment (implemented using Visual Studio shell extensions) and the development repository was written in C++. The environment generated table creation/alteration code and stored procedures for a multitude of databases. Based on the metadata for each Java class, the byte code (generated by Java complier) was 'patched' to insert calls to the specific stored procedures. The clear advantage was code written once in Java could be made to perform differently without recompiling, simply by editing the associated XML. The disadvantage was XML and Java needed to be kept in sync. To overcome this hurdle, we had to develop tools to maintain synchronicities. We could have used C#'s attributes instead of worring about synchronization.

C# introduced attributes, which allow you to embed information right into the C# source code. Attributes are placed in square brackets, e.g. [MTAThread], above an Attribute Target (e.g. class, field, method, etcetera). C# Attributes can be used to mark fields, methods, classes etcetera with markings that another program can interpret. For example you can send instructions to compiler to emit a warning message

```
[Obsolete("This delegate should be avoided. Support may be discontinued in next version")]
```

or generate an error for obsoleted methods.

```
[Obsolete("This method has been obsoleted as of version 1.2.3.4", true)]
```

See help on the [ObsoleteAttribute] in the C# language specification.

Specify attributes between square brackets ([and]) in C++ and between angle brackets (< and >) in Visual Basic.

Examples:

`[STAThreadAttribute]`
`public static int Main(string[] args) {`
`}`
	Indicates that the default threading model for an application is single-threaded apartment. Here the Attribute Target is Method. It also implies that there is a class (implemented by the environment in this case) by the name of STAThreadAttribute, with a parameter-less constructor STAThreadAttribute()

`[MTAThread]`	**[MTAThread] is short for [MTAThreadAttribute], in C#. In other words "Attribute" suffix is optional. If you are working in a mixed environment, you should use the full name MTAThreadAttribute versus the shortcut MTAThread.**
`[Obsolete]`	**[Obsolete] can be used to mark an Attribute Target as obsolete. The compiler picks this attribute and emits a message or generates an error**

`[AttributeUsageAttribute(AttributeTargets.Field)]`
`public class PrecisionAttribute {`
`  public void PrecisionAttribute(int length, int precision) {`
`  }`
`}`
	Indicates that the user defined attribute PrecisionAttribute can be applied to fields only. It also indicates that there is a constructor of name AttributeUsageAttribute(AttributeTargets t) in a class AttributeUsageAttribute. Again this class AttributeUsageAttribute is implemented by the .Net environment.

`[PrecisionAttribute(7, 3)] or` `[Precision(7, 3)]`	**Indicates that there is a constructor of name PrecisionAttribute(int, int) in a class PrecisionAttribute, possibly implemented by you.**
`[In] or [InAttribute]`	**Both mean the same thing.**

`[InAttribute, Out,`
`CustomMarshaller(true,`
`UnmanagedType.U4))]`
`TraderIdentifier trId`
`or`
`[In] [Out]`
`[CustomMarshaller(true,`
`UnmanagedType.U4)]`
`TraderIdentifier trId;`
	Attributes may be combined using the comma separator or listed separately within square brackets.

`[DllImport("User32.dll")]`
`public static extern IntPtr`
`GetWindow(IntPtr hWnd, int uCmd);`
	Instructs the compiler to insert code to allow dynamic access to dll.

Sample Code

Here is a sample that shows how to create a custom attribute and code to use the same program. This simply allows you to store the formatting with the declaration. For example consider these two examples:

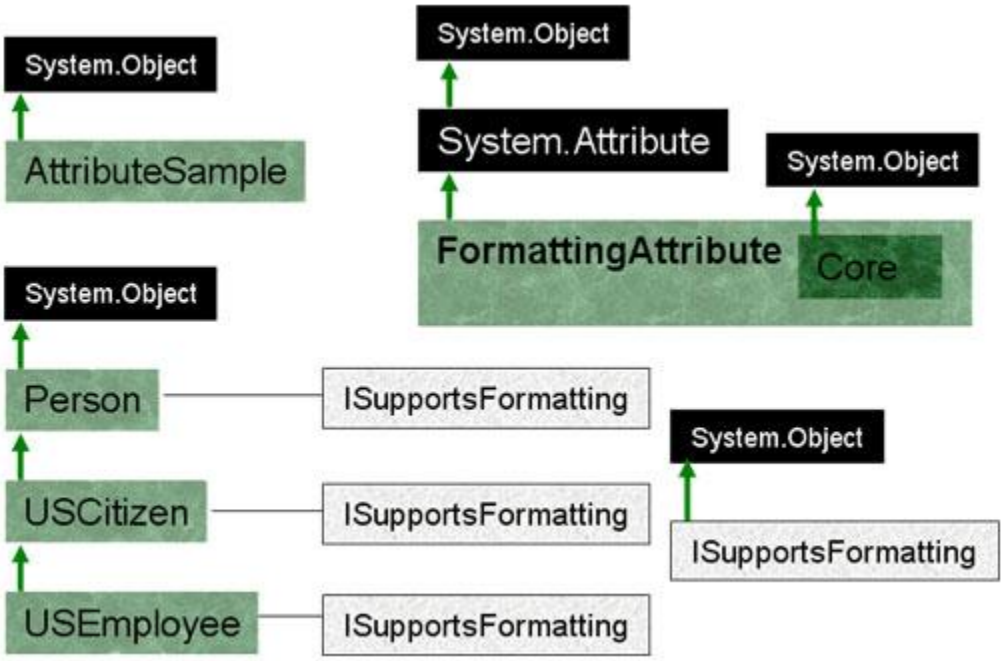
```
[Formatting("###-##-####")]
public int socialSecurityNumber;

[FormattingAttribute("(###) ###-####")]
public long phoneNumber;
```

Class hierarchy

Here is the class hierarchy for the sample code.

Class AttributeSample houses the main function. FormattingAttribute is the code for the attribute and contains a private class Core with core functionality. Classes Person, USCitizen, USEmployee implement the ISupportsFormatting interface and the hierarchical relationship between them is shown in the class hierarchy below. ISupportsFormatting requires methods string Fields() and string Field (string field) to be implemented by the class implementing the interface.



Code

```
using System;
using System.Reflection;
using System.Collections;
using System.Text;

namespace AttributeSample {

    internal class AttributeSample {
        [STAThreadAttribute]
        public static int Main(string[] args) {
            try {
                USEmployee usemployee = new USEmployee("John Doe", 30,
987654321, 8005551212, 36000);

                //Write a specific named field
                Console.WriteLine(usemployee.Field("socialSecurityNumber"));

                Console.WriteLine("");

                //Write all fields
                Console.WriteLine(usemployee.Fields());



              Console.WriteLine("");
Console.WriteLine("");
Console.WriteLine("Paused... Press any key.");

            Console.Read();


            } catch (Exception x){
                Console.WriteLine(x.Message);
                Console.WriteLine(x.StackTrace);
                return -1;
            }
            return 1;
        }
    }


    internal class Person: ISupportsFormatting {
        public string name;

        [FormattingAttribute("### years")]
        public int age;

        [FormattingAttribute("0#/0#/####")]
        public int someImportantDate = 09241972;

        [Formatting("0.###E+000")]
        //You may use 'Formatting' instead of 'FormattingAttribute', in C#.
        //FormattingAttribute is recommended, if you also program in
languages like C++
```

```
        public long someUselessLongValue;

        public Person(string name, int age) {
            someUselessLongValue = -1234567890123456789;
            this.name = name;
            this.age = age;
        }

        public string Fields() {
            return FormattingAttribute.FormattedMembersToString(this);
        }

        public string Field(string field) {
            return FormattingAttribute.FormattedMembers(this, field);
        }

    }

    internal class USCitizen : Person, ISupportsFormatting {
        [FormattingAttribute("###-##-####")]
        public int socialSecurityNumber;

        [FormattingAttribute("(###) ###-####")]
        public long phoneNumber;

        public USCitizen(string name, int age, int socialSecurityNumber, long
phoneNumber) : base(name, age) {
            this.phoneNumber = phoneNumber;
            this.socialSecurityNumber = socialSecurityNumber;
        }

    }

    internal class USEmployee : USCitizen, ISupportsFormatting{

        [FormattingAttribute("00#")]
        public readonly int[] intArray = {1, 17, 9, 141, 12};

        [FormattingAttribute("000.0#")]
        public readonly long[] longArray = {1, 17, 9, 1410008974574645367,
12};

        [FormattingAttribute("$#,#")]
        public int yearlySalary;

        public USEmployee(string name, int age, int ssn, long ph, int sal) :
base(name, age, ssn, ph) {
            yearlySalary = sal;
        }

    }

    internal interface ISupportsFormatting {
        string Fields();
        string Field(string field);
    }
```

```
    [AttributeUsageAttribute(AttributeTargets.Field)]
    internal class FormattingAttribute : System.Attribute {
        public string format;

        public  FormattingAttribute(string format) {
            this.format = format;
        }

        public static string FormattedMembers(object o, string
caseSensitiveFieldName) {
            if (caseSensitiveFieldName == null)
                throw new NullReferenceException();
            return Core.FormattedMembers(o, caseSensitiveFieldName)[0];
        }

        public static string[] FormattedMembers(object o) {
            return Core.FormattedMembers(o, null);
        }

        public static string FormattedMembersToString(object o) {
            string sReturn = null;
            foreach(string s in FormattingAttribute.FormattedMembers(o))
                sReturn += s + delimiter;
            return sReturn;
        }

        private static string delimiter = "\n";
        public static string Delimiter {
            set {
                delimiter = (value == null || value.Length == 0)?"\n":value;
            }
            get {
                return delimiter;
            }
        }

        private class Core {
            public static string[] FormattedMembers(object o, string
caseSensitiveFieldName) {
                if (o == null)
                    throw new NullReferenceException();

                // select the binding flags
                BindingFlags bindingflags = BindingFlags.FlattenHierarchy |
BindingFlags.Public | BindingFlags.Instance;

                //get the member info array. Also look up the Hierarchy
                MemberInfo[] memberInfos =
o.GetType().GetMembers(bindingflags);

                ArrayList a = new ArrayList(memberInfos.GetLength(0));

                //only care for public non-static members in this sample.
                foreach(MemberInfo mi in memberInfos) {

                    //allow overloads to work as intended
```

```
                    if (caseSensitiveFieldName != null &&
caseSensitiveFieldName != mi.Name)
                        continue;

                    //don't care for anything other than fields in this
sample
                    if (mi.MemberType != MemberTypes.Field)
                        continue;

                    //Get the value & type of the field, e.g. 5 in
"[FormattingAttribute("###")] int a = 5;"
                    object field = Field(o, mi.Name);

                    //Get the format e.g. "###" in
"[FormattingAttribute("###")] int a = 5;"
                    string format = Formatting(mi);

                    //now simply write to console or create a string[] to
return
                    a.Add(FieldToLine(field, mi.DeclaringType.ToString(),
mi.Name, format));
                }
                Array strs = Array.CreateInstance(typeof(string), a.Count);
                a.CopyTo(strs);
                a.Clear();

                return (string[])strs;
            }
            private static string FieldToLine(object field, string
declaringClass, string name, string format) {
                //Format the class name
                StringBuilder sb = new StringBuilder("("+declaringClass + ")
");

                if (field == null) {
                    sb.Append(name + " = null");
                    return sb.ToString();
                }

                sb.Append(field.GetType() + " <" + format + "> " + name + " =
");
                if (field.GetType().ToString().EndsWith("[]")) {
                    System.Array array = field as System.Array;
                    if (array == null)
                        sb.Append ("null");
                    else {
                        int len = array.Length;
                        if (len>0)
                            sb.Append("{");
                        for(int i=0; i<len; i++) {
                            sb.Append(FormatObject(array.GetValue(i),
format));
                            sb.Append((i==len-1)?"}":", ");
                        }
                    }
                } else {
                    sb.Append(FormatObject(field, format));
```

```
                }
                return sb.ToString();
            }

            private static string FormatObject(object o, string format) {
                if (typeof(int) == o.GetType())
                    return Convert.ToInt32(o).ToString(format);
                if (typeof(long) == o.GetType())
                    return Convert.ToInt64(o).ToString(format);
                return o.ToString();
            }

            private static object Field(object o, string s) {
                try {
                    return o.GetType().InvokeMember(s, BindingFlags.Instance
|
                        BindingFlags.Public | BindingFlags.GetField, null, o,
new object [] {});
                } catch (Exception x) {
                    Console.WriteLine("Error writing object" + o + " " + s);
                    Console.WriteLine(x.Message);
                    Console.WriteLine(x.StackTrace);
                }
                return null;
            }

            private static string Formatting(MemberInfo mi) {
                //Is there a FormattingAttribute associated with
"[FormattingAttribute("###")] int a = 5;"
                FormattingAttribute a =
(FormattingAttribute)Attribute.GetCustomAttribute(mi,
typeof(FormattingAttribute));

                //Get the format e.g. "###" in "[FormattingAttribute("###")]
int a = 5;"
                if (a != null)
                    return a.format;

                return string.Empty;
            }

        }
    }
}
```

~~~ End of Article ~~~