Obscure C# Coding Features

John F. Gnazzo

Introduction

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is intended to be a simple, modern, general-purpose, object-oriented programming language Its development team is led by Anders Hejlsberg. The most recent version is C# 5.0, which was released on August 15, 2012.

However, there are several hidden features or tricks of C# that even C# fans, addicts, experts barely know including the **modulus(%)** and **null-coalescing(??)** operators**, goto** statement, **DefaultValue** attribute, and the **var** type.

The following sections will define each of these followed by example code which demonstrates each.

Modulus (%) Operator

The Modulus (**%**) operator computes the remainder after dividing its first operand by its second. All numeric types have predefined remainder operators. In the following snippet if myNumber is an evennumber , then it's remainder is 0 when applying the Modulus operator. If myNumber is an odd number, then it's remainder would be 1 when applying the Modulus operator.

```
number % 2
```

Null-coalescing (??) Operator

The null-coalescing operator returns the left-hand operand if the operand is not null; otherwise it returns the right hand operand.

In the following code snippet the variable useNumber is set equal to number if number is NOT null. If number is null, then useNumber is set equal to -99.

```
var useNumber = number ?? -99;
```

Goto Statement

The **goto** statement transfers the program control directly to a labeled statement.

A common use of **goto** is to transfer control to a specific switch-case label or the default label in a switch statement.

The **goto** statement is also useful to get out of deeply nested loops.

In the code snippet below, goto is used to direct the control flow of the program to the even: label if useNumber is an even number and to the odd: label if useNumber is an odd number as determined by the Modulus operation.

```
            if (useNumber % 2 == 0)
```

```
            {
                goto even;
            }

            goto odd;

            even:
                return "You are an EVEN number";

            odd:
                return "You are an ODD number";
```

DefaultValue Attribute

The **DefaultValue** Specifies the default value for a property. It is useful when one wants to have a known default value for the property.

Var Type

Variables that are declared at method scope can have an implicit type **var**. An implicitly typed local variable is strongly typed just as if you had declared the type yourself, but the compiler determines the type. Use of the **var** type makes code easier to read. In the following code snippet the variable _two would be of integer type.

```
var _two = 2;
```

Example

The following code block demonstrates a code block that contains 3 goto statements. In this code block the integer (number) passed in is tested for whether it is null using the Null-coalescing operator (??). If it is null, the **goto myEnd:** statement passes control to the myEnd: label and the function returns *"You passed a null value to this function"* to the calling program.

If not null, the number is tested for eveness using the modulus operator (%). If it has no remainder from the modulus operation, the goto even: statement passes control to the even: label and the function returns "*You are an EVEN number*" to the calling program.

If the number has a remainder from using the modulus operator (%), the goto odd: statement passes control to the odd: label and returns "*You are an ODD number*" to the calling program.

The code block contains a property decorated with the **DefaultValue** Attribute with a value of 2.

```
public class GotoExample
   {
       private int _two = 2;

       [DefaultValue(2)]
       public int MyProperty
       {
           get
           {
               return _two;
           }
           set
```

```
            {
                _two = value;
            }
        }

        public string WhatAmI(int? number)
        {
            var useNumber = number ?? -99;
            if (useNumber == -99) goto myEnd;

            var attributes=TypeDescriptor.GetProperties(this)["MyProperty"].Attributes;
            var att = DefaultValueAttribute)attributes[typeof(DefaultValueAttribute)];

            if (useNumber % (int)att.Value == 0)
            {
                goto even;
            }

            goto odd;

            even:
                return "You are an EVEN number";

            odd:
                return "You are an ODD number";

            myEnd:
                return "You passed a null value to this function";

        }
    }
```

Conclusion

Knowing and understanding obscure C# coding features gives you more tools to solve business problems.