Functions in C#
Executive Summary

Functions—sometimes referred to as subroutines or methods—are an indispensable element of software development. This application note will look at the advantages of using them, how and when to use them, and some examples of how they are used.

Keywords: function, method, sub-routine, program, code, c#, software

Introduction

A function essentially is a portion of code within a larger program that performs a specific task and is relatively independent of the remaining code. Functions can be thought of as little programs by themselves, with an input and an output. In assembly code they are often referred to as “subroutines,” and have been an essential aspect of programming since the first programs were made. Functions are so integral because they can reduce the size and complexity of code by many orders of magnitude. Because of this, the cost of software development is also significantly reduced. Functions are also often collected into libraries and used to share and trade software. Since C# is an Object Oriented Programming (OOP) language, everything in C# is comprised of objects and functions those objects implement.

Components

Functions in C# can take multiple inputs, but can give only one output. Before they can be used, functions must be defined. Defining a function entails giving it a name with which to call it, specifying its input parameters, its body (the actual computational code), the value it will output, and finally its level of accessibility.

Name
- The name of a function is what other functions or objects will call it with. Each function must have a unique name, however functions with different input parameters can have the same name. For example, DoStuff( int number1) and DoStuff( string text1) are two different functions.

Input Parameters
- A function’s input parameters are the initial variables that need to be manipulated. The body of the function will perform some computation with these initial values and yield an output. Not all functions have inputs. Functions without inputs often perform a consistent action or simply call other functions.

Body
- The body of a function is the code which dictates the manipulation of its input variables.

Output
- The output is essentially the result of the function body’s interaction with the function’s input parameters. Not all functions have an output. Often times, a
function uses its inputs to perform an action or trigger an event rather than compute a result.

**Accessibility Level**
- This is a feature that is available only in Object Oriented Programming (OOP) languages. Since C# is an object oriented language, everything is an object, and functions *belong* to an object. Objects are defined as classes, which contain the function definition within their definition. Accessibility for functions is defined by the keywords “private” and “public.” The “private” marker signifies that the function can only be used by the object it is defined under. The “public” marker, however, allows that function to be called by any object that has access to the function’s parent object.

For example, consider a Jeep SUV as an object. This object’s “SpeedUp()” function can only be used by the object itself. If that same function were marked as “public,” other objects (other Jeeps) would be able to control the speed of this Jeep. Obviously, that would lead to chaos! In Programming, the consequences are not as dire, but proper usage of the “public” and “private” markers can make software much more robust.

**Syntax**

```csharp
public int AddNumbers(int number1, int number2)
{
    int result = number1 + number2;
    if(result > 10)
    {
        return result;
    }
    return 0;
}
```

The function’s *declaration* is the first line of its definition, which specifies its level of accessibility, its output type, its name, and its inputs. In this example, this is a public function, which outputs a result of type integer, is called “AddNumbers,” and takes two input parameters both of type integer. In short, this is a function which adds two integers and returns the integer result. The *int* keyword represents the “integer” data type. The *return* keyword specifies what should be output.

The function’s body basically adds the two integers together, and only returns (outputs) the value if it is greater than 10. If the value is not greater than 10, it simply returns 0. Below is an example of how to call this function:

```csharp
int additionResult = AddNumbers(5, 8);
```

“AddNumbers” is called to add 5 and 8, which is 13. Since this is above 10, the result 13 is returned and subsequently stored in the “additionResult” variable.
This is a comprehensive example containing most of the concepts discussed in this document.

The object is defined on the first line, and named “ComputationEngine.” This object contains just 3 functions, two of which are private and one of which is public.

- The first function, “DoMath”, takes no inputs and yields no output. Its sole purpose is to call the other functions. Notice that “DoMath” is a public function; this is because it was intended to be called from outside the owning object. The other two functions are kept private, because they are only called from “DoMath” and so do not need to be accessed from outside the “ComputationEngine” object. The void keyword signifies that nothing is output. Rather, this function triggers the action of writing out the result of the computation for the user to see.

- The second function, “ExecuteFormula”, takes the three numbers initialized in “DoMath” as inputs, and outputs a result based on the given mathematical formula in its code body. It returns a result of type int, and is specified as private because it does not need to be accessed outside the parent class.

- The last function, “PrintNumber”, takes one integer as an input and writes that number on the screen for the user to see. It returns nothing, which is signified by the void keyword.

Essentially, “DoMath” calls “ExecuteFormula”, and then calls “PrintNumber” to print out that result. Organizing code like this makes it easier to read and track, and ensures that only one of our functions is accessible from outside the class and so does not expose any unnecessary information from within other (private) functions for use only within the class.
Conclusion

This application note described what functions are and why they are relevant when creating software. It also discussed some advantages of making use of functions, and included a walk through on how to use them when programming with the C# language. Moreover, this document analyzed the various parts of a function and explained what each part meant. Finally, a comprehensive example collected all of the above ideas and demonstrated their usefulness. To summarize, functions are an indispensable aspect of programming. Proper use of functions can save time, money, and dramatically reduce the learning curve when dealing with a new software development environment. They also are the cornerstone of the software trading world, and allow others’ work to be nicely encapsulated and distributed. While this was just a primer, learning to use functions frequently and appropriately is the key building robust software.

References