platform By Per Scholas

C# Parameter Lists

platform By Per Scholas

Topics Covered

- out Parameters
- ref Parameters
- Variable-length Parameters
- Default (Optional) Parameters
- Named Parameters

C# Parameter Lists

Parameter lists were introduced in the discussion of C# methods. Parameter lists are part of the unique signature of each method or constructor, and they are also an essential part of every delegate definition.

This presentation illustrates some of the variations that are possible with parameter lists, including:

- ref and out parameters
- Variable-length parameter lists
- Default (optional) parameters
- Named parameters

out Parameters

We are familiar with methods that return a value, but C# provides a second mechanism for returning values from a method: out parameters.

Consider the static method double.TryParse:

```
public static bool TryParse (string s, out double result);
```

The documentation for this method states:

> *Converts the string representation of a number to its double-precision floating-point number equivalent. A return value indicates whether the conversion succeeded or failed.*

If TryParse returns true, then the result parameter is set to the parsed value and is useable.

out parameters can be and <u>must be</u> set by the declaring method.

Using out Parameters

In the first example, we declare a local variable d without initialization. We normally cannot use an uninitialized variable, but we can pass it as an out parameter. After the method returns, we are allowed to use d because C# guarantees that out parameters are initialized (if TryParse fails, d is set to zero).

```
static void TryParse()
{
    double d;
    if (double.TryParse("13.8e-4", out d))
    {
        Console.WriteLine($"Value is: {d}");
    }
}
```

The second example shows an implementation identical to the first where d is defined inline.

```
static void TryParse()
{
    if (double.TryParse("13.8e-4", out double d))
    {
        Console.WriteLine($"Value is: {d}");
    }
}
```

Methods with out Parameters

By declaring an out parameter, we are promising that it will be assigned a value before the method returns. The compiler forces us to make that assignment.

Further, we cannot use the out parameter until it has been assigned.

```
static void MethodFour(string name, out double d)
{
    if (DateTime.Now.Second > 30)
    {
        double test = d;  // This is an error - we cannot assume d is initialized.
        return;           // This is an error - we MUST assign d a value before returning.
    }
    d = double.PositiveInfinity;
}
```

Methods can declare multiple out parameters.

ref Parameters

ref parameters combine the effects of “in” (normal) parameters and out parameters.

Unlike out parameters, ref parameters must be initialized before the method call, so the method is free to use them.

Like out parameters, changes made to a ref parameter are available to the calling function.

```
static void MethodFive(string name, ref double d)
{
    if (d > 0) d = Math.PI; else d = Math.E;
}
```

```
// v must be initialized before calling MethodFive
double v = 15.0;
MethodFive(string.Empty, ref v);
Console.WriteLine($"The new value of v is {v}.");
```

ref, out and Overloading

ref or out are part of a method signature, so the first two of these methods are distinct and valid overloads. However, the compiler does not distinguish between ref and out parameters, so the third overload is an error.

```
void MethodX(double d) { }
void MethodX(out double d) { d = 0; }
void MethodX(ref double d) { }
```

void Program.MethodX(ref double d)

'Program' cannot define an overloaded method that differs only on parameter modifiers 'ref' and 'out'

Show potential fixes (Alt+Enter or Ctrl+.)

Variable-length Parameter Lists

A C# method can support an open-ended number of parameters by using the params keyword:

```
static void MethodThree(string name, params double[] values)
{
    Console.WriteLine($"MethodThree called with values: {String.Join(",", values)}");
}
```

```
// Call MethodThree with 0 or more values:
MethodThree("Pete Wilson");
MethodThree("Pete Wilson", 1.0);
MethodThree("Pete Wilson", 1.0, 2.0, 10.0, 500.0);
```

params can be used only once in a method declaration, and it must be the last argument in the list.

Default (Optional) Parameters

Method declarations can define default values for one or more parameters using the following syntax:

```
static void MethodTwo(string name, double value = 10.0, int ival = 100)
{
    Console.WriteLine($"MethodTwo called with value = {value} and ival = {ival}");
}
```

This method declares two default parameters, and can be invoked with any of the following statements:

```
MethodTwo("Pete Wilson");           // use default value and ival
MethodTwo("Pete Wilson", 57.5);     // specify value, use default ival
MethodTwo("Pete Wilson", 57.5, 27);  // specify both value and ival
```

Default parameters must be declared at the end of the parameter list, after all required parameters.

Named Parameters

Most often we call methods with parameters passed by order. C# also gives us the option of passing parameters by name. Given again MethodTwo:

```
static void MethodTwo(string name, double value = 10.0, int ival = 100)
{
    Console.WriteLine($"MethodTwo called with value = {value} and ival = {ival}");
}
```

We can invoke MethodTwo with arguments in any order by naming them:

```
// Call MethodTwo using named parameters:
MethodTwo(value: 27.6, ival: 137, name: "Pete Wilson");
```

platform By Per Scholas

Topics Covered

- out Parameters
- ref Parameters
- Variable-length Parameters
- Default (Optional) Parameters
- Named Parameters