Specifying Clear Method Parameters

Deborah Kurata

@deborahkurata | blogs.msmvps.com/deborahk/

What Does This Do?

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

- `product` - Product we want to order
- `12` - Quantity of that product
- `true` - ???
- `false` - ???

Method Signature & Method Call

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

Module Overview

- Improving Parameters in the Method Signature
- Named Arguments
- Defining Enumerated Parameters
- Optional Parameters
- `ref` and `out` Parameters
- FAQ

Improving Parameters in the Method Signature

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

Coherent Parameter Names

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

XML Document Comment

```
/// <summary>
/// Sends a product order to the vendor.
/// </summary>
/// <param name="product">Product to order.</param>
/// <param name="quantity">Quantity of the product to order.</param>
/// <param name="includeAddress">True to include the shipping address.</param>
/// <param name="sendCopy">True to send a copy of the email.</param>
/// <returns>Success flag and order text</returns>
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
   // Code here
}
```

Limiting the Number of Parameters

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

- 7?
- 4?
- Minimum possible parameters and no more
- Use object type

Logical and Consistent Parameter Order

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

- Acted upon or key to the operation
- Required for the operation
- Flags
- Optional parameters

Method Parameter Best Practices

Do:

Define coherent parameter names

Define an XML document comment for each parameter

Keep the number of parameters to a minimum

Order the parameters in a logical sequence

Use a consistent parameter order

Avoid:

Unused parameters

Current Method Signature & Method Call

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

Named Arguments

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

```
var result = vendor.PlaceOrder(product,
                               quantity: 12,
                               includeAddress: true,
                               sendCopy: false);
```

Named Arguments

```
var result = vendor.PlaceOrder(product,
                               quantity: 12,
                               includeAddress: true,
                               sendCopy: false);
```

- With named arguments, parameter order doesn't matter
- Not all arguments need to be named
- All named arguments must follow positional arguments

Named Argument Best Practices

Do:

Use named arguments as needed for clarity when calling a method

Avoid:

Unnecessary named arguments

```
PlaceOrder(product: product, …
```

Current Method Signature & Method Call

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
{
    // Code here
}
```

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

Avoiding Boolean Parameters

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
```

- Define differing method names
  - PlaceOrder
  - PlaceOrderWithCopy
- Use enum

```
public enum IncludeAddress { Yes, No };
public enum SendCopy { Yes, No };
```

Using enum Parameters

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  bool includeAddress, bool sendCopy)
```

```
public enum IncludeAddress { Yes, No };
public enum SendCopy { Yes, No };
```

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  IncludeAddress includeAddress,
                                  SendCopy sendCopy)
```

```
var result = vendor.PlaceOrder(product, 12, true, false);
```

```
var result = vendor.PlaceOrder(product, 12,
                               Vendor.IncludeAddress.Yes,
                               Vendor.SendCopy.No);
```

enum Parameter Best Practices

Do:

Define a clear name

Use PascalCasing

Use enum to represent a set of related values

Favor enum over a set of constants

Avoid:

Boolean parameters where possible

Consider enum types instead

Using enum for lists that change often

Required Parameters

```
public OperationResult PlaceOrder(Product product, int quantity)

public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy)

public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy, string instructions)
```

Optional Parameters

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy = null,
                                  string instructions = "standard delivery");
```

- Specify a default value
- Are optional when the method is called
- If argument is not provided, default is used
- Can dramatically reduce the number of overloads

Reduces Overloads

```
public OperationResult PlaceOrder(Product product, int quantity)

public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy)

public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy, string instructions)
```

```
public OperationResult PlaceOrder(Product product, int quantity,
                        DateTimeOffset? deliverBy = null,
                        string instructions = "standard delivery");
```

Optional Parameters

```
public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy = null,
                                  string instructions = "standard delivery")
```

- Optional parameters must be defined after required parameters
- When calling the method, if an argument is provided for any optional parameter, it must also provide arguments for all preceding parameters
  - Or use named arguments

Optional Parameter Best Practices

Do:

Use optional parameters to minimize overload bloat

Avoid:

Optional parameters when the parameters are one or the other

```
FindProduct(int id);
FindProduct(string productName);
FindProduct(int id=0,
            string productName="");
```

Optional parameters if default could change and component versioning is important

Returning Multiple Values

```
public OperationResult PlaceOrder(Product product, int quantity,
                        DateTimeOffset? deliverBy = null,
                        string instructions = "standard delivery")
{
  // Code here
  var operationResult = new OperationResult(success, orderText);
  return operationResult;
}
```

ref and out Parameters

```
public bool PlaceOrder(Product product, int quantity,
                       ref string orderText)
```

```
public bool PlaceOrder(Product product, int quantity,
                       out string orderText)
```

- Returns multiple values:
  - Value defined in the return statement
  - One or more parameter values defined with the ref or out keyword

By Value

```
var orderText = "Standard Order";
var actual = vendor.PlaceOrder(product, 12, orderText);
Console.WriteLine(orderText);
```

```
public bool PlaceOrder(Product product, int quantity,
                       string orderText)
{
    var success = true;
    orderText = "Order from Acme";
    return success;
}
```

By Reference

```
var orderText = "Standard Order";
var actual = vendor.PlaceOrder(product, 12, ref orderText);
Console.WriteLine(orderText);
```

```
public bool PlaceOrder(Product product, int quantity,
                                        ref string orderText)
{
    var success = true;
    orderText = "Order from Acme";
    return success;
}
```

`ref` vs. `out`

`ref`

- Argument passed "by reference"
- Argument variable must be initialized
- Parameter value can be changed in the method
- Changes are reflected in the calling code

`out`

- Argument passed "by reference"
- Argument variable must be declared
- Parameter value must be set in the method
- Changes are reflected in the calling code

ref and out Parameters

```
public bool PlaceOrder(Product product, int quantity,
                       ref string orderText)
```

```
public bool PlaceOrder(Product product, int quantity,
                       out string orderText)
```

```
public OperationResult PlaceOrder(Product product, int quantity)
{
  // Code here
  var operationResult = new OperationResult(success, orderText);
  return operationResult;
}
```

`ref` and `out` Best Practices

Do:

Use ref when the method expects an incoming value

Use `out` when the method expects no incoming value

Avoid:

`ref` and `out` where feasible

Return an object instead

Frequently Asked Questions

- What is the difference between a parameter and an argument?
  - A parameter is part of the method signature
  - An argument is part of the method call
- What is a named argument and when should it be used?
  - A named argument uses the parameter name when calling the method
  - Used to clarify the purpose of an argument and define arguments without concern for their position in the parameter list
- How is an optional parameter defined?
  - By specifying a default value

Frequently Asked Questions (cont)

- What is the difference between passing an argument by value vs. by reference?
  - When passed by value (which is the default), the value of the argument is passed to the method
  - When passed by reference (using ref or out), the variable is effectively passed to the method
  - Because of this, passing by reference enables the method to change the value of the parameter and have that change reflected in the calling code

Frequently Asked Questions (cont)

- What is the difference between `ref` and `out`?
  - A `ref` parameter requires that the argument be initialized before it is passed.
    - The method can modify the value for the `ref` parameter.
  - An `out` parameter must be declared, but not initialized before it is passed.
    - The method **must** provide a value for the `out` parameter.

This Module Covered

- Improving Parameters in the Method Signature
- Named Arguments
- Defining Enumerated Parameters
- Optional Parameters
- `ref` and `out` Parameters