## **12.5 The Generic Action<> and Func<> Delegates**

**Over the course of this chapter, you have seen that when you want to use delegates to enable callbacks in your applications, you typically follow the steps shown here:**

1. **Define a custom delegate that matches the format of the method being pointed to.**
2. **Create an instance of your custom delegate, passing in a method name as a constructor argument.**
3. **Invoke the method indirectly, via a call to Invoke() on the delegate object.**

**In many cases, you simply want “some delegate” that takes a set of arguments and possibly has a return value other than void. In these cases, you can use the framework’s built-in Action<> and Func<> delegate types. To illustrate their usefulness, create a new Console Application project named ActionAndFuncDelegates. The generic Action<> delegate is defined in the System namespace, and you can use this generic delegate to “point to” a method that takes up to 16 arguments (that ought to be enough!) and returns void. Now recall, because Action<> is a generic delegate, you will need to specify the underlying types of each parameter as well.**

**Exampe:**

**// This is a target for the Action<> delegate.**

**static void DisplayMessage(string msg, ConsoleColor txtColor, int printCount)**

**{**

**// Set color of console text.**

**ConsoleColor previous = Console.ForegroundColor;**

**Console.ForegroundColor = txtColor;**

**}**

**Now, rather than building a custom delegate manually to pass the program’s flow to the**

**DisplayMessage() method, you can use the out-of-the-box Action<> delegate, as so:**

**Console.WriteLine("\*\*\*\*\* Fun with Action and Func \*\*\*\*\*");**

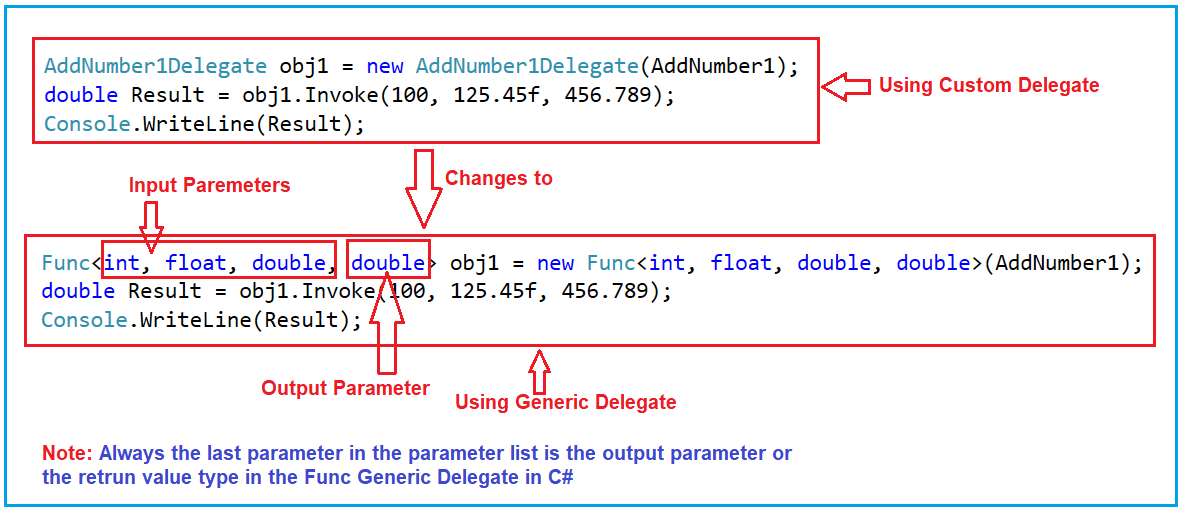
**// Use the Action<> delegate to point to DisplayMessage.**

**Action<string, ConsoleColor, int> actionTarget = DisplayMessage;**

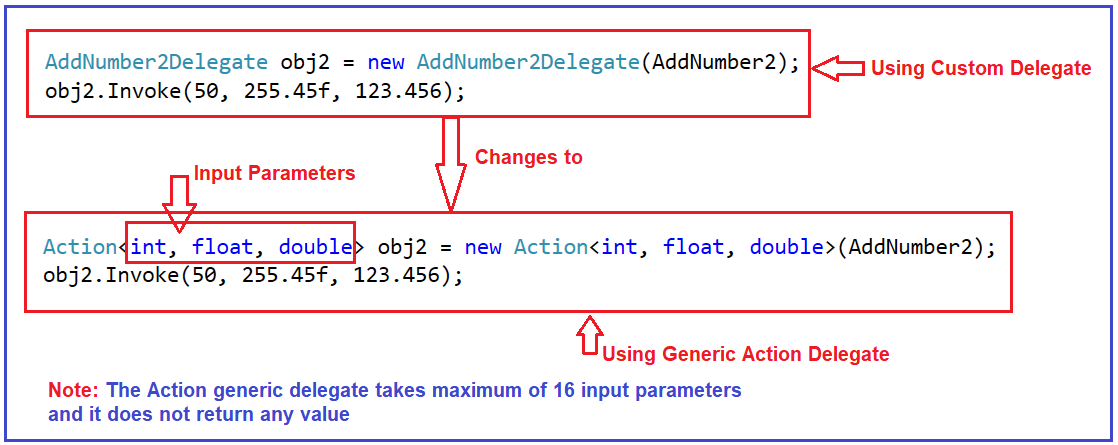
**actionTarget("Action Message!", ConsoleColor.Yellow, 5);**

**Console.ReadLine();**

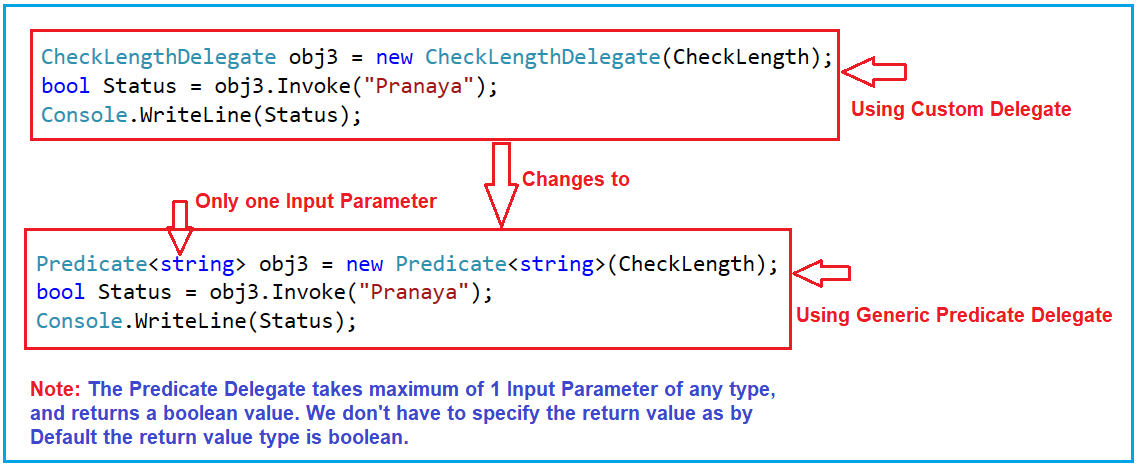
The Func Generic Delegate takes params and returns one parm (the last one)



The Action Generic Delegate takes parms and return none.



The Predicate Generic Delegate takes parms and returns bool



## C# Predicate

Predicates in C# are implemented with delegates. The Predicate delegate represents the method that defines a set of criteria and determines whether the specified object meets those criteria.

The following example creates a simple C# Predicate.

**Program.cs**

var data = new List<int> { 1, -2, 3, 0, 2, -1 };

var predicate = new Predicate<int>(isPositive);

var filtered = data.FindAll(predicate);

Console.WriteLine(string.Join(",", filtered));

bool isPositive(int val)

{

return val > 0;

}

## C# predicate with Func

The Func is a generic delegate type. It can contain 0 to 16 input parameters and must have one return type. Predicate is a specialization of Func.

**Program.cs**

var data = new List<Person>

{

new ("John Doe", "gardener"),

new ("Robert Brown", "programmer"),

new ("Lucia Smith", "teacher"),

new ("Thomas Neuwirth", "teacher")

};

ShowOutput(data, r => r.Occupation == "teacher");

void ShowOutput(List<Person> list, Func<Person, bool> condition)

{

var data = list.Where(condition);

foreach (var person in data)

{

Console.WriteLine($"{person.Name}, {person.Occupation}");

}

}

record Person(string Name, string Occupation);

Notes: