Different Ways To Create Delegates In C#

**Introduction**

A Delegate is a reference type that means it encapsulates method(s). It **decouples** the class that declares the delegate from the class that uses it. Declaring class doesn't need to know what class will use the Delegate.

Delegates are very useful in some cases like passing method as parameter, callback, calling multiple methods in a single event, etc. But, this article is not intended to explain detailed concepts of Delegate. It will only explain the different ways to create a Delegate.

The following are the different ways to create or use Delegate:

1. Action
2. Func
3. Predicate
4. Lambda
5. Anonymous Types.

Basically, all these types of Delegate provide a way to create a generic delegate without explicitly declaring a custom delegate. I will explain in detail about all these kinds of Delegates.

I will explain and focus on only Action, Func and Predicate Delegates.

**Action**

An Action is a type of Delegate. Basically, it encapsulates a method that has a single parameter and does not return a value. It is required when developer doesn't want to declare Delegate explicitly.

Some important points about Action:

1. It doesn't return any value.
2. It may take zero to 16 parameters.
3. It can be used with anonymous methods or lambda expressions.

**Basic example**

1. **static** **void** Main(**string**[] args)
2. {
3. Action<**int**> myAction = (**int** x) => Console.WriteLine("Print {0}", x);
4. Action<**int**, **int**> myAction1 = (x, y) => Console.WriteLine("Print {0} and {1}", x, y);
6. //Execute Method
7. myAction.Invoke(10);
8. myAction1.Invoke(20, 10);
9. Console.ReadKey();
10. }

**Output**

Print 10

Print 20 and 10

**Action for the short event handlers**

1. mybtnOk.Click += (sender, e) => MessageBox.Show("You have clicked to  save!");

**Action can be used with "Console.WriteLine"**

1. **string** myStringValue = "India,USA,UAE,Canada,Singapur,Switzerland";
2. **string**[] countries = myStringValue.Split(',');
4. // Passing WriteLine as the action
5. Array.ForEach(countries, Console.WriteLine);
7. Console.ReadKey();

India

USA

UAE

Canada

Switzerland

**Anonymous methods can be used with Action**

1. **static** **void** Main(**string**[] args)
2. {
3. Action<**int**> showValue = **delegate**(**int** i) { Console.WriteLine(i); };
4. showValue(10);
6. Console.ReadKey();
7. }

**Output**

10

**Func**

Func is also a kind of Delegate that encapsulates a method that has parameter(s) and returns a value of the type specified by the TResult parameter.

Some important points about Func:

1. It is used to point to a method that returns a value.
2. It takes up to 16 parameters and returns a value.

The following are the common usage examples of Func Delegate:

**Func is being used in LINQ - Projection**

1. **static** **void** Main(**string**[] args)
2. {
3. List<Student> students = **new** List<Student>();
4. students.Add(**new** Student { RollNo = 1, Name = "ABC" });
5. students.Add(**new** Student { RollNo = 2, Name = "ABC" });
6. students.Add(**new** Student { RollNo = 3, Name = "ABC" });
7. var studentDetails = students.Select(x => x.RollNo);
8. **foreach** (var item **in** studentDetails)
9. Console.WriteLine(item);
10. Console.ReadKey();
11. }

**Output**

1

2

3

**Func is being used in LINQ - Filtering**

1. **static** **void** Main(**string**[] args)
2. {
3. List<Student> students = **new** List<Student>();
4. students.Add(**new** Student { RollNo = 1, Name = "Ali" });
5. students.Add(**new** Student { RollNo = 2, Name = "Rohan" });
6. students.Add(**new** Student { RollNo = 3, Name = "Jon" });
7. var studentDetails = students.Where(x => x.RollNo == 1);
8. **foreach** (var item **in** studentDetails)
9. Console.WriteLine(item.Name);
10. Console.ReadKey();
11. }

Output: Ali

**Func can be used as Key Selection**

The following example will co-relate the elements of two sequences based on key.

1. **public** **class** Employee
2. {
3. **public** **int** EmployeeId { **get**; **set**; }
4. **public** **string** Name { **get**; **set**; }
5. }
7. **public** **class** EmployeeSalary
8. {
9. **public** Double Salary { **get**; **set**; }
10. **public** Employee EmployeeDetail { **get**; **set**; }
11. }
12. **static** **void** Main(**string**[] args)
13. {
14. Employee employeeAli = **new** Employee { EmployeeId = 1, Name = "Ali" };
15. Employee employeeRaj = **new** Employee { EmployeeId = 2, Name = "Raj" };
17. EmployeeSalary empSalaryAli = **new** EmployeeSalary { Salary = 2000, EmployeeDetail = employeeAli };
18. EmployeeSalary empSalaryRaj = **new** EmployeeSalary { Salary = 80000, EmployeeDetail = employeeRaj };
20. List<Employee> employee = **new** List<Employee> { employeeAli, employeeRaj };
21. List<EmployeeSalary> employeeSalary = **new** List<EmployeeSalary> { empSalaryAli, empSalaryRaj };
23. var employeeDetailsQuery = employee.Join(employeeSalary, emp => emp, sal => sal.EmployeeDetail,
24. (emp, sal) => **new** { EmployeeName = emp.Name, Salary = sal.Salary });
26. **foreach** (var employeeDetail **in** employeeDetailsQuery)
27. {
28. Console.WriteLine("{0} - {1}", employeeDetail.EmployeeName, employeeDetail.Salary);
29. }
31. Console.ReadKey();
32. }

Here, Join has been used as Func.

Output:

Ali - 2000

Raj - 80000

**Func can be used to project a sequence of Integer values into a sequence of strings**

1. **static** **void** Main(**string**[] args)
2. {
4. // Here, It is a projection.
5. Func<**int**, **string**> myValueDetails = intData => "Value : " + intData;
6. **int**[] integerValues = { 1, 2, 3, 4 };
7. var strings = integerValues.Select(myValueDetails);
9. **foreach** (**string** s **in** strings)
10. {
11. Console.WriteLine(s);
12. }
13. Console.ReadKey();
14. }

Output:

Value : 1

Value : 2

Value : 3

Value : 4

So, in the above code you can see integer value has been projected with string value by using Func.

**Predicate**

Predicate is a kind of Delegate that represents the method, defines a set of criteria and test whether it is true of given T object.

It is basically reference to a function that return true or false.

Predicate, Predicate<T> is similar to Func<T, bool>. It is very useful and allows us to write logic at run time. Most common usage of predicate are - Filtering a list of values, searching items in a collection, etc.

The following are the examples where Predicate Delegate has been used.

**Filter a number**

1. **public** **static** **bool** IsEvenNumber(**int** number)
2. {
3. **return** number % 2 == 0;
4. }
5. **static** **void** Main(**string**[] args)
6. {
8. List<**int**> numbers = **new** List<**int**> { 1, 2, 3, 8,11,20 };
9. Predicate<**int**> myPredicate = **new** Predicate<**int**>(IsEvenNumber);
10. List<**int**> evenNumbers = numbers.FindAll(myPredicate);
11. **foreach** (var even **in** evenNumbers)
12. {
13. Console.WriteLine(even);
14. }
16. Console.ReadKey();
17. }

Output:

2

8

20

**But, C# 3.0 provides a great way to use Predicate. See like the following:**

1. **static** **void** Main(**string**[] args)
2. {
3. List<**int**> numbers = **new** List<**int**> { 1, 2, 3, 4, 8, 11 };
4. List<**int**> evenNumbers = numbers.FindAll(numb => numb % 2 == 0);
5. **foreach** (var evenNum **in** evenNumbers)
6. Console.WriteLine(evenNum);
7. Console.ReadKey();
8. }

Output:

2

4

8

**Conclusion**

C# framework provided us a wonderful way to use or define Delegate in the form of Action, Predicate, Func, etc. This provide generic way to create Delegate without explicitly declaring a custom Delegate. It saves our time and effort by not defining our own Delegate types for the same thing.