Delegates

Delegates are used to invoke the call-back functions. What it means is we will invoke one function and we will pass the delegate instance as a parameter to that function and we expect that function to invoke the delegate at some point of time which will invoke the callback method referred by the delegate instance.

Delegates Overview

Delegates have the following properties:

- Delegates are similar to C++ function pointers, but delegates are fully object-oriented, and unlike C++ pointers to member functions, delegates encapsulate both an object instance and a method.
- Delegates allow methods to be passed as parameters.
- Delegates can be used to define callback methods.
- Delegates can be chained together; for example, multiple methods can be called on a single event.
- Methods don't have to match the delegate type exactly. For more information, see Using Variance in Delegates.
- Lambda expressions are a more concise way of writing inline code blocks. Lambda expressions (in certain contexts) are compiled to delegate types. For more information about lambda expressions, see Lambda expressions.
- A delegate is a type that represents references to methods with a particular parameter list and return type.
- When you instantiate a delegate, you can associate its instance with any method with a compatible signature and return type.
- You can invoke (or call) the method through the delegate instance.
- Delegates are used to pass methods as arguments to other methods.
- Event handlers are nothing more than methods that are invoked through delegates.

Any method from any accessible class or struct that matches the delegate type can be assigned to the delegate. The method can be either static or an instance method. This flexibility means you can programmatically change method calls, or plug new code into existing classes.

Note

In the context of method overloading, the signature of a method does not include the return value. But in the context of delegates, the signature does include the return value. In other words, a method must have the same return type as the delegate.

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods.

```
[access modifier] delegate [return type] [delegate name]([parameters])
```

Benefits and drawbacks of delegates in C#

Here are a few benefits of delegates to keep in mind:

- Delegates can be used to invoke static and non-static methods.
- A delegate can be used to call one or more methods having identical signatures.
- Delegates can be used to define callback methods and invoke event handlers.
- Delegates can be combined into multicast delegates that execute a chain of delegates in order.

Here are some of the downsides of using delegates:

- A delegate will execute more slowly than a conventional method because the runtime needs to resolve the method reference before the delegate call is resolved and the method is called.
- An exception thrown in a method referenced by a delegate doesn’t show the reason for the error, which makes debugging more difficult.
- The more delegates you use, the less readable your code becomes.
- When you use delegates, the JIT compiler and runtime may optimize far less than conventional functions.

Delegates are ideally suited to implementing event-driven programming. A delegate doesn’t need to know the class of the object to which it refers. All a delegate needs to know is the signature of the method to which it would point.

Beginning with C# 10, method groups with a single overload have a *natural type*. This means the compiler can infer the return type and parameter types for the delegate type.

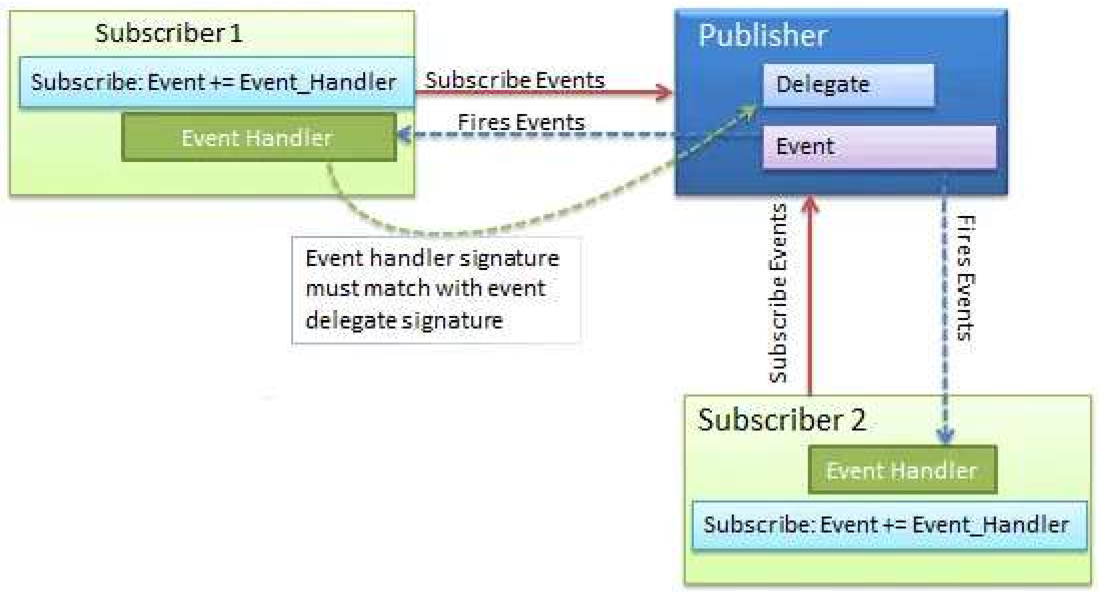
Event in C#?

Events in C#, being a subset of delegates are defined by using delegates. To raise an event in C# you need a publisher, and to receive and handle an event you need a subscriber or multiple subscribers.

These are usually implemented as publisher and subscriber classes.

Events in C#, being a subset of delegates are defined by using… delegates. To raise an event in C# you need a publisher, and to receive and handle an event you need a subscriber or multiple subscribers.

These are usually implemented as publisher and subscriber classes. Event delegates typically have two parameters. The first one is for the source, or rather the class that will be publishing the event, and the second one of the type EventArgs, which is any additional data related to the event.


Subscriber 1
Subscribe: Event += Event_Handler
Event Handler
Subscribe Events
Fires Events
Publisher
Delegate
Event
Event handler signature must match with event delegate signature
Subscribe Events
Fires Events
Subscriber 2
Event Handler
Subscribe: Event += Event_Handler

Anonymous Method

As the name suggests, an anonymous method is a method without a name. Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.

Anonymous methods can also be passed to a method that accepts the delegate as a parameter.

```
public delegate void Print(int value);

class Program
{
    public static void PrintHelperMethod(Print printDel,int val)
    {
        val += 10;
        printDel(val);
    }

    static void Main(string[] args)
    {
        PrintHelperMethod(delegate(int              val)              {
Console.WriteLine("Anonymous method: {0}", val); }, 100);
    }
}
```

Extension Methods

- Extension methods enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type.
- Extension methods are static methods, but they're called as if they were instance methods on the extended type. For client code written in C#, F# and Visual Basic,

there's no apparent difference between calling an extension method and the methods defined in a type.

- The most common extension methods are the LINQ standard query operators that add query functionality to the existing System.Collections.IEnumerable and System.Collections.Generic.IEnumerable<T> types.

- Extension methods are defined as static methods but are called by using instance method syntax. Their first parameter specifies which type the method operates on.
- The parameter is preceded by the this modifier. Extension methods are only in scope when you explicitly import the namespace into your source code with a `using` directive.