**Events** are user actions such as key press, clicks, mouse movements, etc., or some occurrence such as system generated notifications. Applications need to respond to events when they occur. For example, interrupts. Events are used for inter-process communication.

Using Delegates with Events

The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class. The class containing the event is used to publish the event. This is called the **publisher** class. Some other class that accepts this event is called the **subscriber**class. Events use the **publisher-subscriber** model.



A **publisher** is an object that contains the definition of the event and the delegate. The event-delegate association is also defined in this object. A publisher class object invokes the event and it is notified to other objects.

A **subscriber** is an object that accepts the event and provides an event handler. The delegate in the publisher class invokes the method (event handler) of the subscriber class.



Declaring Events

To declare an event inside a class, first of all, you must declare a delegate type for the even as:

public delegate string BoilerLogHandler(string str);

then, declare the event using the **event** keyword −

event BoilerLogHandler BoilerEventLog;

The preceding code defines a delegate named *BoilerLogHandler* and an event named *BoilerEventLog*, which invokes the delegate when it is raised.

Example

using System;

namespace SampleApp {

public delegate string MyDel(string str);

class EventProgram {

event MyDel MyEvent;

public EventProgram() {

this.MyEvent += new MyDel(this.WelcomeUser);

}

public string WelcomeUser(string username) {

return "Welcome " + username;

}

static void Main(string[] args) {

EventProgram obj1 = new EventProgram();

string result = obj1.MyEvent("Tutorials Point");

Console.WriteLine(result);

}

}

}

When the above code is compiled and executed, it produces the following result −

Welcome Tutorials Point

1. **public** **class** MyTest
2. {
3. **public** **event** EventHandler MyEvent
4. {
5. add
6. {
7. Console.WriteLine("add operation");
8. }
9. remove
10. {
11. Console.WriteLine("remove operation");
12. }
13. }
14. }
15. **public** **class** Test
16. {
17. **public** **void** TestEvent()
18. {
19. MyTest myTest = **new** MyTest();
20. myTest.MyEvent += myTest\_MyEvent;
21. myTest.MyEvent -= myTest\_MyEvent;
22. }
23. **public** **void** myTest\_MyEvent(**object** sender, EventArgs e)
24. {
25. }
26. }
27. **static** **void** Main(**string**[] args)
28. {
29. Test test = **new** Test();
30. test.TestEvent();
31. Console.ReadKey();
32. }

### WHAT IS EVENTS?

Events are nothing just a user action. For example

1. When you click with the mouse – It is mouse click events.
2. When you press any key in keyboard – It is KeyPress events
3. When you refresh your webpage – It is page load events
4. When you move mouse cursor – It is mouse hover events etc.

So when you take any action like a key press, mouse movements, clicks etc an event raised. Let me clear more about it. For example, you filled an online form and click on submit button.

1. In the background **button\_click()** event raised.
2. This event calls and execute an associated function **Submit\_Click()**.
3. This function processes your request and submits page information to database.

### HOW EVENTS WORK WITH DELEGATES?

Delegates are used to reference a method. An Event is associated with Event Handler using Delegates. When an Event raise, it sends a signal to delegates and delegates executes the right function.

### WHAT IS PUBLISHER-SUBSCRIBER MODEL?

There are two parts in any event handling program. One part is Publisher that contains definition of events and delegates and another part is Subscriber that accepts the event and provides an event handler.

### IMPORTANT FACT ABOUT EVENTS

1. An Event is created using event keyword.
2. An Event has no return type and it is always void.
3. All events are based on delegates.
4. All the published events must have a listening object.
5. All Events should be defined starting with “On” keyword.

Let's understand all these theory using Programming Example  
Before seeing the programming examples you must know the sequential steps to manipulate events.  
  
**Step 1:** Define a Delegate  
**Step 2:** Define an Event with same name of Delegates.  
**Step 3:** Define an Event Handler that respond when event raised.  
**Step 4:** You must have method ready for delegates.

### PROGRAMMING EXAMPLE:

This program simply adds two numbers. Only condition is if the sum of number is odd it fires an event that print a message using delegates.

1. using **System**;
2. using **System**.**Collections**.**Generic**;
3. using **System**.**Linq**;
4. using **System**.**Text**;
5. using **System**.**Threading**.**Tasks**;
7. namespace event\_programming
8. {
9. //This is Subscriber Class
10. class **Program**
11. {
12. static void **Main**(string[] args)
13. {
14. **AddTwoNumbers** a = new **AddTwoNumbers**();
15. //Event gets binded with delegates
16. a.ev\_OddNumber += new **AddTwoNumbers**.dg\_OddNumber(**EventMessage**);
17. a.**Add**();
18. **Console**.**Read**();
19. }
20. //Delegates calls this method when event raised.
21. static void **EventMessage**()
22. {
23. **Console**.**WriteLine**("\*\*\*\*\*\*\*\*Event Executed : This is Odd Number\*\*\*\*\*\*\*\*\*\*");
24. }
25. }
26. //This is Publisher Class
27. class **AddTwoNumbers**
28. {
29. public delegate void dg\_OddNumber(); //Declared Delegate
30. public event dg\_OddNumber ev\_OddNumber; //Declared Events
32. public void **Add**()
33. {
34. int result;
35. result = 5 + 4;
36. **Console**.**WriteLine**(result.**ToString**());
37. //Check if result is odd number then raise event
38. if((result % 2 != 0) && (ev\_OddNumber != null))
39. {
40. ev\_OddNumber(); //Raised Event
41. }
42. }
43. }
44. }