**Events**

* Another important C# feature is built upon the foundation of delegates: the event.
* An event is, essentially, an automatic notification that some action has occurred.
* Events work like this: An object that has an interest in an event registers an event handler for that event.
* When the event occurs, all registered handlers are called.
* Event handlers are represented by delegates.
* Events are members of a class and are declared using the event keyword.
* Its most commonly used form is shown here:

event event-delegate event-name;

* Here, event-delegate is the name of the delegate used to support the event, and event-name is the name of the specific event object being declared.
* Let’s begin with a very simple example:

// A very simple event demonstration.

**using System;**

// Declare a delegate type for an event.

**delegate void MyEventHandler();**

// Declare a class that contains an event.

**class MyEvent**

{

**public event MyEventHandlerSomeEvent;**

// This is called to fire the event.

**public void OnSomeEvent()**

{

if(SomeEvent != null)

SomeEvent();

}

}

**class EventDemo**

{

// An event handler.

**static void Handler()**

{

Console.WriteLine("Event occurred");

}

**public static void Main(String[] args)**

{

**MyEventevt = new MyEvent();**

// Add Handler() to the event list.

**evt**.SomeEvent += Handler;

// Fire the event.

**evt**.OnSomeEvent();

}

}

**This program displays the following output:**

Event occurred

* Although simple, this program contains all the elements essential to proper event handling.
* Let’s look at it carefully. The program begins by declaring a delegate type for the event handler, as shown here:
  + **delegate void MyEventHandler();**
* All events are activated through a delegate. Thus, the event delegate type defines the return type and signature for the event.
* In this case, there are no parameters, but event parameters are allowed.
* Next, an event class, called MyEvent, is created. Inside the class, an event called SomeEvent is declared, using this line:
  + **public event MyEventHandlerSomeEvent;**
* Notice the syntax. The keyword event tells the compiler that an event is being declared. Also declared inside MyEvent is the method OnSomeEvent( ), which is the method a program will call to signal (or “fire”) an event. (That is, this is the method called when the event occurs.)
* It calls an event handler through the SomeEvent delegate, as shown here:

**if(SomeEvent != null)**

**SomeEvent();**

* Notice that a handler is called if and only if SomeEvent is not null. Since other parts of your program must register an interest in an event in order to receive event notifications, it is possible that OnSomeEvent( ) could be called before any event handler has been registered.
* To prevent calling on a null reference, the event delegate must be tested to ensure that it is not null.
* Inside EventDemo, an event handler called Handler( ) is created. In this simple example, the event handler simply displays a message, but other handlers could perform more meaningful actions.
* In Main( ), a MyEvent object is created, and Handler( ) is registered as a handler for this event, by adding it as shown here:

**MyEventevt = new MyEvent();**

**// Add Handler() to the event list.**

**evt.SomeEvent += Handler;**

* Notice that the handler is added using the += operator.
* Events support only += and – =.
* In this case, Handler( ) is a static method, but event handlers can also be instance methods.
* Finally, the event is fired as shown here:

**// Fire the event.**

**evt.OnSomeEvent();**

* Calling OnSomeEvent( ) causes all registered event handlers to be called. In this case, there is only one registered handler, but there could be more, as the next section explains.