The **event** keyword lets you specify a delegate that will be called upon the occurrence of some "event" in your code. The delegate can have one or more associated methods that will be called when your code indicates that the event has occurred. An event in one program can be made available to other programs that target the .NET Framework common language runtime.

The following steps must be taken in order to create and use C# events:

1. Create or identify a delegate. If you are defining your own event, you must also ensure that there is a delegate to use with the event keyword. If the event is predefined, in the .NET Framework for example, then consumers of the event need only know the name of the delegate.
2. Create a class that contains:
   1. An event created from the delegate.
   2. (optional) A method that verifies that an instance of the delegate declared with the **event** keyword exists. Otherwise, this logic must be placed in the code that fires the event.
   3. Methods that call the event. These methods can be overrides of some base class functionality.

This class defines the event.

1. Define one or more classes that connect methods to the event. Each of these classes will include:
   1. Associate one or more methods, using the += and -= operators, with the event in the base class.
   2. The definition of the method(s) that will be associated with the event.
2. Use the event:
   1. Create an object of the class that contains the event declaration.
   2. Create an object of the class that contains the event definition, using the constructor that you defined.

For more information on events, see:

* [10.7 Events](https://msdn.microsoft.com/en-us/library/aa664454%28v=vs.71%29.aspx)
* [Events Tutorial](https://msdn.microsoft.com/en-us/library/aa288460%28v=vs.71%29.aspx)
* [Events and Delegates](https://msdn.microsoft.com/en-us/library/17sde2xt%28v=vs.71%29.aspx)

**Examples**

A basic example of defining and using events can be found in the [Events Tutorial](https://msdn.microsoft.com/en-us/library/aa645739%28v=vs.71%29.aspx). The following examples in this topic also demonstrate events.

|  |  |
| --- | --- |
| **Example** | **Comment** |
| [Example 1](https://msdn.microsoft.com/en-us/library/8627sbea%28v=vs.71%29.aspx#vclrfeventpgexample1) | Declaring an event in an interface and implementing it in a class. |
| [Example 2](https://msdn.microsoft.com/en-us/library/8627sbea%28v=vs.71%29.aspx#vclrfeventpgexample2) | Using a hash table to store event instances. |
| [Example 3](https://msdn.microsoft.com/en-us/library/8627sbea%28v=vs.71%29.aspx#vclrfeventpgexample3) | Implementing, via event properties, two interfaces that have an event with the same name. |

**Example 1**

This example shows that it is possible to declare an event in an interface and implement it in a class:

// event\_keyword.cs

using System;

public delegate void MyDelegate(); // delegate declaration

public interface I

{

event MyDelegate MyEvent;

void FireAway();

}

public class MyClass: I

{

public event MyDelegate MyEvent;

public void FireAway()

{

if (MyEvent != null)

MyEvent();

}

}

public class MainClass

{

static private void f()

{

Console.WriteLine("This is called when the event fires.");

}

static public void Main ()

{

I i = new MyClass();

i.MyEvent += new MyDelegate(f);

i.FireAway();

}

}

**Example 2**

One use for *accessor-declarations* is to expose a large number of events without allocating a field for each event, but instead using a hash table to store the event instances. This is only useful if you have a very large number of events, but you expect most of the events will not be implemented.

// event\_keyword2.cs

using System;

using System.Collections;

public delegate void MyDelegate1(int i);

public delegate void MyDelegate2(string s);

public delegate void MyDelegate3(int i, object o);

public delegate void MyDelegate4();

public class PropertyEventsSample

{

private Hashtable eventTable = new Hashtable();

public event MyDelegate1 Event1

{

add

{

eventTable["Event1"] = (MyDelegate1)eventTable["Event1"] + value;

}

remove

{

eventTable["Event1"] = (MyDelegate1)eventTable["Event1"] - value;

}

}

public event MyDelegate1 Event2

{

add

{

eventTable["Event2"] = (MyDelegate1)eventTable["Event2"] + value;

}

remove

{

eventTable["Event2"] = (MyDelegate1)eventTable["Event2"] - value;

}

}

public event MyDelegate2 Event3

{

add

{

eventTable["Event3"] = (MyDelegate2)eventTable["Event3"] + value;

}

remove

{

eventTable["Event3"] = (MyDelegate2)eventTable["Event3"] - value;

}

}

public event MyDelegate3 Event4

{

add

{

eventTable["Event4"] = (MyDelegate3)eventTable["Event4"] + value;

}

remove

{

eventTable["Event4"] = (MyDelegate3)eventTable["Event4"] - value;

}

}

public event MyDelegate3 Event5

{

add

{

eventTable["Event5"] = (MyDelegate3)eventTable["Event5"] + value;

}

remove

{

eventTable["Event5"] = (MyDelegate3)eventTable["Event5"] - value;

}

}

public event MyDelegate4 Event6

{

add

{

eventTable["Event6"] = (MyDelegate4)eventTable["Event6"] + value;

}

remove

{

eventTable["Event6"] = (MyDelegate4)eventTable["Event6"] - value;

}

}

}

public class MyClass

{

public static void Main()

{

}

}

**Example 3**

Another use for event properties covers the situation where you are implementing two interfaces, each with an event of the same name. In such a case, you must use an explicit implementation event property:

// event\_keyword3.cs

using System;

public delegate void MyDelegate1();

public interface I1

{

event MyDelegate1 MyEvent;

}

public delegate int MyDelegate2(string s);

public interface I2

{

event MyDelegate2 MyEvent;

}

public class ExplicitEventsSample: I1, I2

{

public event MyDelegate1 MyEvent; // normal implementation of I1.MyEvent.

event MyDelegate2 I2.MyEvent // explicit implementation of I2.MyEvent

{

add

{

MyEvent2Storage += value;

}

remove

{

MyEvent2Storage -= value;

}

}

private MyDelegate2 MyEvent2Storage; // underlying storage for I2.MyEvent.

private void FireEvents()

{

if (MyEvent != null)

MyEvent();

if (MyEvent2Storage != null)

MyEvent2Storage("hello");

}

}

public class MyClass

{

public static void Main()

{

}

}