

C# - Event

In general terms, an event is something special that is going to happen. For example, Microsoft launches events for developers, to make them aware about the features of new or existing products. Microsoft notifies the developers about the event by email or other advertisement options. So in this case, Microsoft is a publisher who launches (raises) an **event** and **notifies** the developers about it and developers are the **subscribers** of the event and attend (**handle**) the event.

Events in C# follow a similar concept. An event has a publisher, subscriber, notification and a handler. Generally, UI controls use events extensively. For example, the button control in a Windows form has multiple events such as click, mouseover, etc. A custom class can also have an event to notify other subscriber classes about something that has happened or is going to happen. Let's see how you can define an event and notify other classes that have event handlers.

An event is nothing but an encapsulated delegate. As we have learned in the previous section, a delegate is a reference type data type. You can declare the delegate as shown below:

Example: Delegate

```
public delegate void someEvent();  
public someEvent someEvent;
```

Now, to declare an event, use the **event** keyword before declaring a variable of delegate type, as below:

Example: Event Declaration

```
public delegate void someEvent();  
public event someEvent someEvent;
```

Thus, a delegate becomes an event using the **event** keyword.

Now, let's see a practical example of an event. Consider the following PrintHelper class that prints integers in different formats like number, money, decimal, temperature and hexadecimal. It includes a beforePrintEvent to notify the subscriber of the BeforePrint event before it going to print the number.

Example: Event

```
public class PrintHelper
{
    // declare delegate
    public delegate void BeforePrint();

    //declare event of type delegate
    public event BeforePrint beforePrintEvent;

    public PrintHelper()
    {
    }

    public void PrintNumber(int num)
    {
        //call delegate method before going to print
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Decimal: {0:G}", dec);
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Money: {0:C}", money);
    }

    public void PrintTemperature(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Temperature: {0,4:N1} F", num);
    }

    public void PrintHexadecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent();

        Console.WriteLine("Hexadecimal: {0:X}", dec);
    }
}
```

P
r
i



The delegate can also be invoked using the Invoke() method, e.g., beforePrintEvent.Invoke().

PrintHelper is a publisher class that publishes the beforePrint event. Notice that in each print method, it first checks to see if beforePrintEvent is not null and then it calls *beforePrintEvent()*. beforePrintEvent is an object of type BeforPrint delegate, so it would be null if no class is subscribed to the event and that is why it is necessary to check for null before calling a delegate.

Now, let's create a subscriber. Consider the following simple Number class for example.

Example: Event subscriber

```
class Number
{
    private PrintHelper _printHelper;

    public Number(int val)
    {
        _value = val;

        _printHelper = new PrintHelper();
        //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }
    //beforePrintevent handler
    void printHelper_beforePrintEvent()
    {
        Console.WriteLine("BeforPrintEventHandler: PrintHelper is going to print
a value");
    }

    private int _value;

    public int Value
    {
        get { return _value; }
        set { _value = value; }
    }

    public void PrintMoney()
    {
        _printHelper.PrintMoney(_value);
    }

    public void PrintNumber()
    {
        _printHelper.PrintNumber(_value);
    }
}
```

Try it

All the subscribers must provided a handler function, which is going to be called when a publisher raises an event. In the above example, the Number class creates an instance of PrintHelper and subscribes to the beforePrintEvent with the "+=" operator and gives the name of the function which will handle the event (it will be get called when publish fires an event). *printHelper_beforePrintEvent* is the event handler that has the same signature as the BeforePrint delegate in the PrintHelper class.

So now, create an instance of Number class and call print methods:

Example: Event

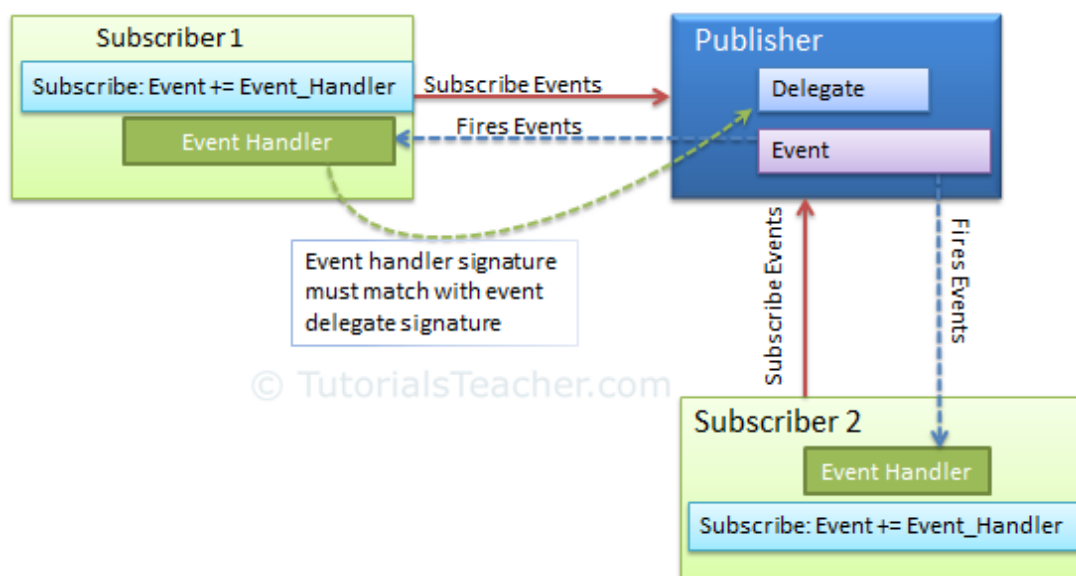
```
Number myNumber = new Number(100000);  
myNumber.PrintMoney();  
myNumber.PrintNumber();
```

Try it

Output:

```
BeforePrintEventHandler: PrintHelper is going to print  
value  
Money: $ 1,00,000.00  
BeforePrintEventHandler: PrintHelper is going to print  
value  
Number: 1,00,000
```

The following image illustrates an event model:



Event publisher-Subscriber

Event Arguments

Events can also pass data as an argument to their subscribed handler. An event passes arguments to the handler as per the delegate signature. In the following example, PrintHelper declares the BeforePrint delegate that accepts a string argument. So now, you can pass a string when you raise an event from PrintNumber or any other Print method.

Example: Event Arguments

```
public class PrintHelper
{
    public delegate void BeforePrint(string message);
    public event BeforePrint beforePrintEvent;

    public void PrintNumber(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent.Invoke("PrintNumber");

        Console.WriteLine("Number: {0,-12:N0}", num);
    }

    public void PrintDecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintDecimal");

        Console.WriteLine("Decimal: {0:G}", dec);
    }

    public void PrintMoney(int money)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintMoney");

        Console.WriteLine("Money: {0:C}", money);
    }

    public void PrintTemperature(int num)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintTemerature");

        Console.WriteLine("Temperature: {0,4:N1} F", num);
    }

    public void PrintHexadecimal(int dec)
    {
        if (beforePrintEvent != null)
            beforePrintEvent("PrintHexadecimal");

        Console.WriteLine("Hexadecimal: {0:X}", dec);
    }
}
```

Now, the subscriber class should have an event handler that has a string parameter.

In the following example, Number class has a `printHelper_beforePrintEvent` function with string parameter.

Example: Event

```
class Number
{
    private PrintHelper _printHelper;

    public Number(int val)
    {
        _value = val;

        _printHelper = new PrintHelper();
        //subscribe to beforePrintEvent event
        _printHelper.beforePrintEvent += printHelper_beforePrintEvent;
    }
    //beforePrintevent handler
    void printHelper_beforePrintEvent(string message)
    {
        Console.WriteLine("BeforePrintEvent fires from {0}",message);
    }

    private int _value;

    public int Value
    {
        get { return _value; }
        set { _value = value; }
    }

    public void PrintMoney()
    {
        _printHelper.PrintMoney(_value);
    }

    public void PrintNumber()
    {
        _printHelper.PrintNumber(_value);
    }
}
```

Try it

Output:

```
BeforePrintEvent fires from PrintMoney.
Money: $ 1,00,000.00
BeforePrintEvent fires from PrintNumber.
Number: 1,00,000
```

Further Reading



Points to Remember :

Use event keyword with delegate type to declare an event.

Check event is null or not before raising an event.

Subscribe to events using "+=" operator. Unsubscribe it using "-=" operator.

Function that handles the event is called event handler. Event handler must have same signature as declared by event delegate.

Events can have arguments which will be passed to handler function.

Events can also be declared static, virtual, sealed and abstract.

An Interface can include event as a member.

Events will not be raised if there is no subscriber

Event handlers are invoked synchronously if there are multiple subscribers

The .NET framework uses an [EventHandler](#) delegate and an [EventArgs](#) base class.

Share

Tweet

Share

Whatsapp