**实现接口事件（C# 编程指南）**

[接口](https://msdn.microsoft.com/zh-cn/library/87d83y5b.aspx)可声明[事件](https://msdn.microsoft.com/zh-cn/library/8627sbea.aspx)。下面的示例演示如何在类中实现接口事件。实现接口事件的规则与实现任何接口方法或属性的规则基本相同。

**在类中实现接口事件**

* 在类中声明事件，然后在适当的区域调用该事件。
* namespace ImplementInterfaceEvents
* {
* public interface IDrawingObject
* {
* event EventHandler ShapeChanged;
* }
* public class MyEventArgs : EventArgs
* {
* // class members
* }
* public class Shape : IDrawingObject
* {
* public event EventHandler ShapeChanged;
* void ChangeShape()
* {
* // Do something here before the event…
* OnShapeChanged(new MyEventArgs(/\*arguments\*/));
* // or do something here after the event.
* }
* protected virtual void OnShapeChanged(MyEventArgs e)
* {
* if(ShapeChanged != null)
* {
* ShapeChanged(this, e);
* }
* }
* }
* }

下面的示例演示如何处理以下的不常见情况：您的类是从两个以上的接口继承的，每个接口都含有同名事件）。在这种情况下，您至少要为其中一个事件提供显式接口实现。为事件编写显式接口实现时，必须编写 **add** 和 **remove** 事件访问器。这两个事件访问器通常由编译器提供，但在这种情况下编译器不能提供。

您可以提供自己的访问器，以便指定这两个事件是由您的类中的同一事件表示，还是由不同事件表示。例如，根据接口规范，如果事件应在不同时间引发，则可以将每个事件与类中的一个单独实现关联。在下面的示例中，订户将形状引用强制转换为 IShape 或 IDrawingObject，从而确定自己将会接收哪个 OnDraw 事件。

namespace WrapTwoInterfaceEvents

{

using System;

public interface IDrawingObject

{

// Raise this event before drawing

// the object.

event EventHandler OnDraw;

}

public interface IShape

{

// Raise this event after drawing

// the shape.

event EventHandler OnDraw;

}

// Base class event publisher inherits two

// interfaces, each with an OnDraw event

public class Shape : IDrawingObject, IShape

{

// Create an event for each interface event

event EventHandler PreDrawEvent;

event EventHandler PostDrawEvent;

object objectLock = new Object();

// Explicit interface implementation required.

// Associate IDrawingObject's event with

// PreDrawEvent

event EventHandler IDrawingObject.OnDraw

{

add

{

lock (objectLock)

{

PreDrawEvent += value;

}

}

remove

{

lock (objectLock)

{

PreDrawEvent -= value;

}

}

}

// Explicit interface implementation required.

// Associate IShape's event with

// PostDrawEvent

event EventHandler IShape.OnDraw

{

add

{

lock (objectLock)

{

PostDrawEvent += value;

}

}

remove

{

lock (objectLock)

{

PostDrawEvent -= value;

}

}

}

// For the sake of simplicity this one method

// implements both interfaces.

public void Draw()

{

// Raise IDrawingObject's event before the object is drawn.

EventHandler handler = PreDrawEvent;

if (handler != null)

{

handler(this, new EventArgs());

}

Console.WriteLine("Drawing a shape.");

// RaiseIShape's event after the object is drawn.

handler = PostDrawEvent;

if (handler != null)

{

handler(this, new EventArgs());

}

}

}

public class Subscriber1

{

// References the shape object as an IDrawingObject

public Subscriber1(Shape shape)

{

IDrawingObject d = (IDrawingObject)shape;

d.OnDraw += new EventHandler(d\_OnDraw);

}

void d\_OnDraw(object sender, EventArgs e)

{

Console.WriteLine("Sub1 receives the IDrawingObject event.");

}

}

// References the shape object as an IShape

public class Subscriber2

{

public Subscriber2(Shape shape)

{

IShape d = (IShape)shape;

d.OnDraw += new EventHandler(d\_OnDraw);

}

void d\_OnDraw(object sender, EventArgs e)

{

Console.WriteLine("Sub2 receives the IShape event.");

}

}

public class Program

{

static void Main(string[] args)

{

Shape shape = new Shape();

Subscriber1 sub = new Subscriber1(shape);

Subscriber2 sub2 = new Subscriber2(shape);

shape.Draw();

// Keep the console window open in debug mode.

System.Console.WriteLine("Press any key to exit.");

System.Console.ReadKey();

}

}

}

/\* Output:

Sub1 receives the IDrawingObject event.

Drawing a shape.

Sub2 receives the IShape event.

\*/