

EVENTOS EN NET FRAMEWORK

Un *evento* en C# es el modo que tiene una clase de proporcionar notificaciones a los clientes de la clase cuando ocurre algo digno de reseñar en un objeto. El uso más habitual para los eventos se produce en las interfaces gráficas; normalmente, las clases que representan controles de la interfaz disponen de eventos que se notifican cuando el usuario hace algo con el control (por ejemplo, hacer clic en un botón).

Los eventos, sin embargo, no sólo se utilizan para interfaces gráficas. Los eventos proporcionan un medio apropiado para que los objetos puedan señalar cambios de estado que pueden resultar útiles para los clientes de ese objeto. Los eventos constituyen unidades importantes para crear clases que se pueden reutilizar en diferentes programas.

Cuando ocurre algo interesante, los eventos habilitan una clase u objeto para notificarlo a otras clases u objetos. La clase que envía (o *genera*) el evento recibe el nombre de *publicador* y las clases que reciben (o *controlan*) el evento se denominan *suscriptores*.

En una aplicación web típica en C#, se puede suscribir a eventos generados por controles, como botones y cuadros de lista. Puede usar el entorno de desarrollo integrado (IDE) de Visual C# para examinar los eventos que publica un control y seleccionar los que quiera administrar. El IDE agrega automáticamente un método de controlador de eventos vacío y el código para suscribirse al evento.

Los eventos tienen las siguientes propiedades:

- El publicador determina el momento en el que se genera un evento; los suscriptores determinan la acción que se lleva a cabo en respuesta al evento.
- Un evento puede tener varios suscriptores. Un suscriptor puede controlar varios eventos de varios publicadores.
- Nunca se generan eventos que no tienen suscriptores.
- Los eventos se suelen usar para indicar acciones del usuario, como los clics de los botones o las selecciones de menú en las interfaces gráficas de usuario.
- Cuando un evento tiene varios suscriptores, los controladores de eventos se invocan sincrónicamente cuando se genera un evento.
- En la biblioteca de clases .NET Framework, los eventos se basan en el delegado **EventHandler** y en la clase base **EventArgs**.

Suscribirse a eventos desde el IDE de Visual Studio

Las siguientes imágenes y ejemplos de código, corresponden a una aplicación Asp.Net con Webforms.

Si no puede ver la ventana **Propiedades**, en la vista **Diseño** haga clic con el botón secundario del mouse en el formulario o control para el que desea crear un controlador de eventos y seleccione **Propiedades**.



Se desplegará la ventana de **Propiedades**.



En la parte superior de la ventana **Propiedades**, haga clic en el icono **Eventos**.



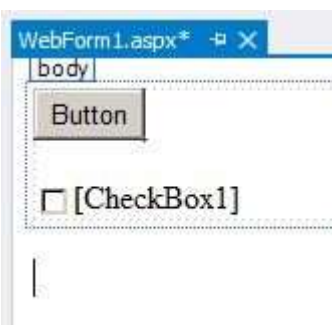
Haga doble clic en el evento que desea crear (por ejemplo, el evento **Click** de un botón). Visual C# crea un método de control de eventos vacío y lo agrega al código. También puede agregar manualmente el código en la vista **Código**. Por ejemplo, el siguiente código declara un método de control de eventos al que se llamará cuando la clase **Button** genere el evento **Click**.

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Aquí agregar el Código para responder al evento click del botón
}
```

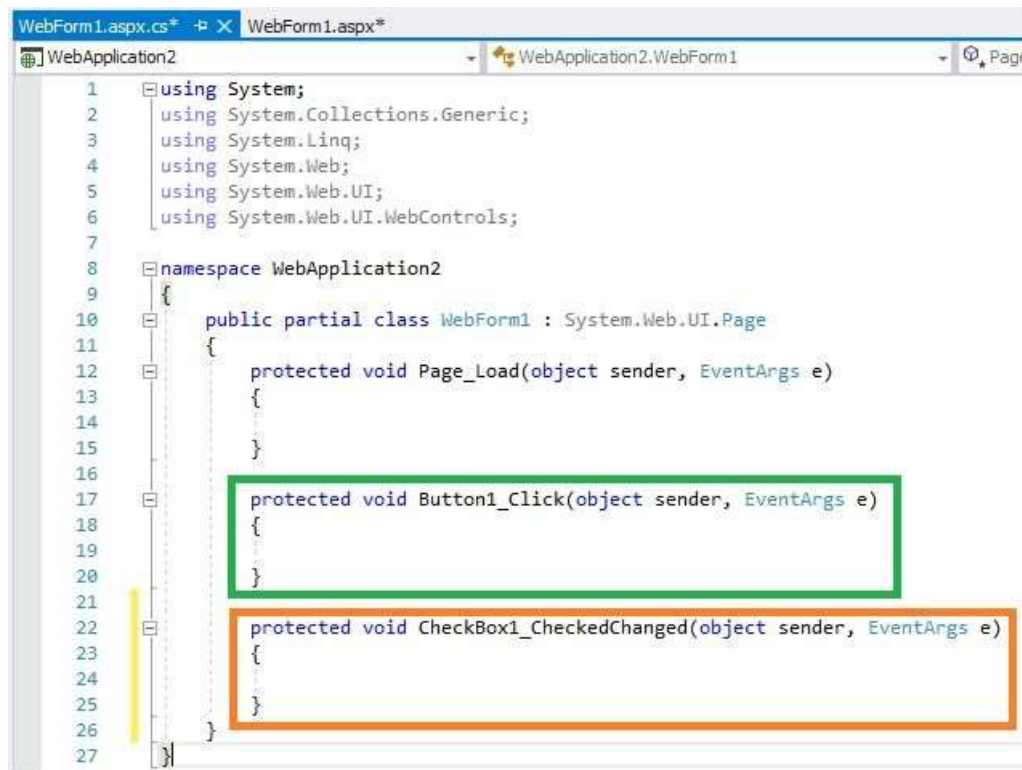
Cuando se efectúa en un Webform un doble clic sobre un Control o el mismo formulario web, genera automáticamente una suscripción de evento al evento por omisión de la clase (en el ejemplo anterior el evento por omisión de la clase **Button** es **Click**), y genera de manera automática el método necesario para su manejo en el código fuente.

Cada control posee sus propios eventos por omisión aunque compartan la mayoría de ellos.

Ejemplos:



Note que tanto el **Button** como el **CheckBox** en forma predeterminada poseen eventos diferentes:



```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.UI;
6 using System.Web.UI.WebControls;
7
8 namespace WebApplication2
9 {
10     public partial class WebForm1 : System.Web.UI.Page
11     {
12         protected void Page_Load(object sender, EventArgs e)
13         {
14         }
15
16         protected void Button1_Click(object sender, EventArgs e)
17         {
18         }
19
20         protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
21         {
22         }
23
24     }
25 }
26
27
```

Además de ver los manejadores de evento por omisión de las clases **Button** y **CheckBox**, vemos el manejador de evento **Page_Load** que corresponde al evento de carga (Load) en un formulario web.

Suscribirse a eventos mediante programación

Suscribirse a un evento significa tener un manejador asociado al mismo que ejecutará el método indicado al dispararse el evento.

Defina un método de control de eventos cuya firma coincida con la firma de delegado del evento. Por ejemplo, si el evento se basa en el tipo de delegado EventHandler, el siguiente código representa el código auxiliar del método:

```
void ManejarEventoPersonalizado(object sender, CustomEventArgs a)
{
    // Hacer algo útil.
}
```

Utilice el operador de suma y asignación (+=) para asociar el controlador de eventos al evento. En el siguiente ejemplo, se asume que un objeto denominado publisher tiene un evento denominado RaiseCustomEvent. Observe que la clase de suscriptor necesita una referencia a la clase de editor para suscribirse a sus eventos.

```
publisher.RaiseCustomEvent += HandleCustomEvent;
```

El delegado encapsulador debe crearse explícitamente mediante la palabra clave `new`:

```
publisher.RaiseCustomEvent += new CustomEventHandler(HandleCustomEvent);
```

También puede agregarse un controlador de eventos utilizando una expresión lambda

```
// Con una expresion lambda se define un evento personalizado.
```

```
this.Click += (s, e) => { MessageBox.Show(  
    ((MouseEventArgs) e).Location.ToString());  
};
```

Parámetro EventArgs

Representa la **clase base** para las clases que contienen datos de eventos, y proporciona un valor que se utilizará para eventos que no incluyen datos de eventos. **Posee la información del evento que se dispara.**

Parámetro Sender

Este parámetro posee información acerca del objeto que disparó el evento. O sea si por ejemplo en el caso de un **Button** se disparó un evento **Click**, entonces tendrá información del objeto **Button** que lo disparó. Recuerde que puede enlazar múltiples controles a un mismo método, por esto **Sender** tiene información del control que disparó el mismo.

Suscribirse a eventos mediante un método anónimo

Veremos más detalles de objetos anónimos más adelante en este curso.

- Si no tiene que cancelar la suscripción a un evento, puede utilizar el operador de suma y asignación (`+=`) para asociar un método anónimo al evento (método sin nombre). En el siguiente ejemplo, se asume que un objeto denominado **publisher** tiene un evento denominado **RaiseCustomEvent** y que se ha definido una clase **CustomEventArgs** para proporcionar algún tipo de información específica del evento. Observe que la clase de suscriptor necesita una referencia a **publisher** para suscribirse a sus eventos.

```
publisher.RaiseCustomEvent += delegate ( object o, CustomEventArgs e)
{
    string s = o.ToString() + " " + e.ToString();
    Console.WriteLine(s);
};
```

- Es importante tener en cuenta que puede no resultar fácil cancelar la suscripción a un evento si se ha utilizado una función anónima para suscribirse a él. Para cancelar la suscripción en esta situación, es necesario regresar al código donde se ha suscrito al evento, almacenar el método anónimo en una variable de delegado y, a continuación, agregar el delegado al evento. En general, se recomienda que no utilice funciones anónimas para suscribirse a eventos si va a tener que cancelar la suscripción al evento en el código más adelante.

Cancelar una suscripción

Para impedir que se invoque el controlador de eventos cuando se produce el evento, basta con cancelar la suscripción al evento. Para evitar que se pierdan recursos, debe cancelar la suscripción a los eventos antes de eliminar un objeto suscriptor. Hasta que se cancela la suscripción a un evento, el delegado multidifusión subyacente al evento en el objeto de publicación tiene una referencia al delegado que encapsula el controlador de eventos del suscriptor. Mientras el objeto de publicación mantenga esa referencia, la recolección de elementos no utilizados no eliminará el objeto suscriptor.

Utilice el operador de resta y asignación (-=) para cancelar la suscripción a un evento:

```
publisher.RaiseCustomEvent -= HandleCustomEvent;
```

Cuando se haya cancelado la suscripción a un evento de todos los suscriptores, la instancia del evento en la clase de editor se establecerá en null.