**0. Introducción**

Empiezo esta nueva serie de artículos (que serán 3, precisamente) en donde tendremos la oportunidad de entrar en nuevo tema: **eventos** (*events*, en inglés). Veremos las posibilidades que brindan a nuestras aplicaciones que requieran un modelo basado en **publicador-subscritor**. En esta primera parte vamos a hacer una introducción general para poder entender y aplicar el concepto de evento. En la segunda nos enfocaremos en el patrón estándar de eventos, y en la última parte, estudiaremos eventos modificadores y eventos de consulta (*getters* o *accessors*).

**1. Eventos**

**1.1 Concepto Genérico**

Un **evento** consiste en una acción generada a partir de la interacción con una aplicación: clic de *mouse*, pulsación de tecla o botón, selección de comando de un menú, arrastrar y soltar elementos en pantalla, &c. Los eventos también puede generarse por la generación de la actividad de un sistema información: por ejemplo en un sistema de cajeras automáticos (ATM), una vez que un cliente de la entidad bancaria realiza una transacción, el sistema se encarga de hacer un registro de la actividad para controlar y asegurar los valores de los clientes de la entidad, igualmente, el mismo sistema puede enviar un mensaje de texto al móvil del cliente con el registro (lugar, valor, fecha y hora) de la transacción recién efectuada. (En la Figura 1 se puede apreciar pictóricamente todos los *eventos* anteriores.)

|  |
| --- |
| [Eventos en un sistema informático cotidiano](http://3.bp.blogspot.com/-It17dnV4oiA/U4-0dVvkBaI/AAAAAAAAIJM/XnoS603swdA/s1600/Art%C3%ADculo+-+20140604+-+Eventos+-+Parte+1+-+Eventos+en+un+sistema+inform%C3%A1tico+cotidiano.png) |
| Figura 1. Eventos en un sistema informático cotidiano. |

**1.2 Concepto en C#**

Básicamente, un *evento* en C# es un mecanismo que permite la generación de notificaciones a un conjunto de *clientes* cuando algún suceso interesante transcurra sobre un objeto. Sucesos a los que podríamos distinguir como interesantes (ejemplos):

* Cambio del estado del objeto.
* Clic sobre un botón de envío de información de un formulario.
* Descarga de archivo completada.
* Notificación sobre con un globo de texto cuando la sincronización de un archivo se ha completado.
* Porcentaje del avance de un proceso sobre una barra de progreso.
* Cambio del nombre del archivo afecta el texto de la barra de título de una aplicación.

Todos los anteriores, y muchos otros *eventos* son generados a partir de una clase *publicadora* y notificados a un conjunto de clases clientes (o *subscriptores*) que están interesados en la generación de esos cambios producidos por los eventos.

En la programación de interfaces gráfica de usuario es común el uso de eventos. Los eventos se asocian con controles (componentes gráficos: botones, listas, cajas de comprobación, áreas de texto, agrupador de elementos, &c.). Cada vez que un usuario pulsa un botón, por ejemplo, se *dispara* un evento que genera una acción particular (guardar archivo, cerrar el programa, ir a la siguiente página, etc.).

|  |
| --- |
| [Eventos en una interfaz gráfica de usuario](http://4.bp.blogspot.com/-BrBqM1tcrNQ/U4_AlZOOkOI/AAAAAAAAIJc/z0-VshzoBms/s1600/Art%C3%ADculo+-+20140604+-+Eventos+-+Parte+1+-+Eventos+en+una+interfaz+gr%C3%A1fica+de+usuario.png) |
| Figura 2. Eventos en una interfaz gráfica de usuario. |

Aparte de la utilidad anterior de los eventos, éstos también son usados para cualquier escenario que requiera la notificación de cambios de estados de un objeto a un conjunto de clientes interesados en conocer su estado y reaccionar frente a esos cambios.  
  
Por otra parte, en la serie de artículos «Delegados en C#» aprendimos a declarar, usar y crear programas usando delegados ([Delegados en C# - Parte 1: Introducción](http://ortizol.blogspot.com/2013/10/namespaces-en-c-parte-1.html)). Saber en qué consiste delegados nos permitirá avanzar más fácilmente en esta serie de «Eventos en C#», debido a que un evento es un tipo de delegado *especial* (por especificarlo de algún modo).  
  
Más técnicamente, un evento es un mecanismo utilizado por una clase para exponer una serie de delegados de métodos, los cuales son invocados una vez ocurra el evento asociado a esa clase de objeto.

**1.3 Palabra clave event**

La palabra clave event se usa para hacer que una clase sea tratada como la generadora de eventos. Se usa de la siguiente manera:

modificador-acceso **event** TipoDelegado identificador;

Descripción puntual:

* modificador-acceso: cualquiera de los modificadores disponibles en C# ([Modificadores de acceso en C#](http://ortizol.blogspot.com/2014/04/modificadores-de-acceso-en-c.html)).
* event: palabra reservada de C#.
* TipoDelegado: tipo de delegado de la biblioteca de clases de Microsoft .NET Framework o uno definido por el programador.
* identificador: nombre o identificador del evento.

Ejemplo específico:

Declaramos un delegado:

**public** **delegate** **void** **CalcularEventHandler**(**int** a, **int** b);

Especificamos un evento a partir de **CalcularEventHandler** (*EventHandler* es un postfijo para aludir que es un delegado diseñado para un evento):

**public** **event** CalcularEventHandler calculo;

**1.3.1 Ejemplo concreto**

En el siguiente ejemplo vamos a crear un delegado compatible con métodos de asignación de números. Luego crearemos un evento asociado al delegado previo, para notificar los cambios que se hagan sobre el campo de una clase *publicadora*.

|  |
| --- |
| { |
| public class Broadcaster |
| { |
| CambioPrecioEventHandler \_cambioPrecio; // Delegado privado |
|  |
| public event CambioPrecioEventHandler CambioPrecio |
| { |
| add |
| { |
| \_cambioPrecio += value; |
| } |
| remove |
| { |
| \_cambioPrecio += value; |
| } |
| } |
|  |
| public event CambioPrecioEventHandler CambioPrecio; |
| } |
|  |
| internal class EjemploEvento |
| { |
| // Campo que siempre que sea modificado se |
| // notificará su nuevo estado a través de un evento: |
| private int valor; |
|  |
| // Delegado diseñado para el evento que 'monitorea' |
| // el cambio de estado del campo `valor`: |
| public delegate void ManipulacionCampoValorEventHandler(); |
|  |
| // Evento que monitorea el estado del campo `valor`: |
| public event ManipulacionCampoValorEventHandler CambioNumero; |
|  |
| // Constructor que inicializa el campo `numero`: |
| public EjemploEvento(int numero) |
| { |
| EstablecerValor (numero); |
| } |
|  |
| protected virtual void OnValorCambio() |
| { |
| if (CambioNumero != null) |
| { |
| CambioNumero(); |
| } |
| else |
| { |
| Console.WriteLine("Evento: El estado del campo `valor` ha cambiado."); |
| } |
| } |
|  |
| public void EstablecerValor(int numero) |
| { |
| if (valor != numero) |
| { |
| valor = numero; |
| OnValorCambio(); |
| } |
| } |
| } |
|  |
| internal class Aplicacion |
| { |
| public static void Main() |
| { |
| // Inclusive cuando creamos una instancia de `EjemploEvento` |
| // se genera el evento: |
| EjemploEvento e = new EjemploEvento(5); |
|  |
| // Otras operaciones que generan un evento: |
| e.EstablecerValor(7); |
| e.EstablecerValor(11); |
| } |
| } |