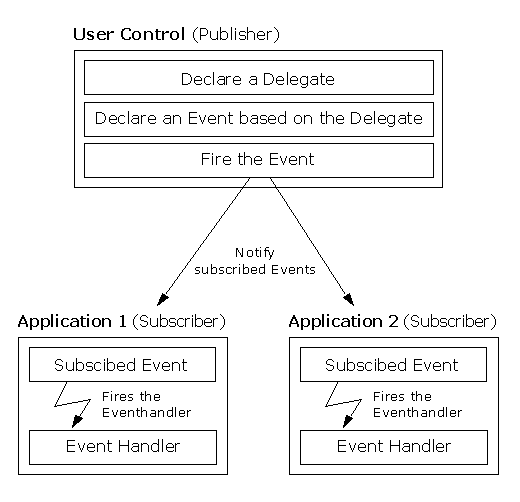
**Events in C# / .NET**

**Prerequisites:** understanding of Delegates and multicast Delegates in C#

In this model, you have publishers who will do some logic and publish an "event." Publishers will then send out their event only to subscribers who have subscribed to receive the specific event.

In C#, any object can publish a set of events to which other applications can subscribe. When the publishing class raises an event, all the subscribed applications are notified. The following figure shows this mechanism.

****

**Simplest Example possible on Events and Delegates in C#:**

using System;

public class Publisher //main publisher class which will invoke methods of all subscriber classes

{

public delegate void TickHandler(Publisher m, EventArgs e); //declaring a delegate

public TickHandler Tick; //creating an object of delegate

public EventArgs e = null; //set 2nd paramter empty

public void Start() //starting point of thread

{

while (true)

{

System.Threading.Thread.Sleep(300);

if (Tick != null) //check if delegate object points to any listener classes method

{

Tick(this, e); //if it points i.e. not null then invoke that method!

}

}

}

}

public class Subscriber1 //1st subscriber class

{

public void Subscribe(Publisher m) //get the object of pubisher class

{

m.Tick += HeardIt; //attach listener class method to publisher class delegate object

}

private void HeardIt(Publisher m, EventArgs e) //subscriber class method

{

System.Console.WriteLine("Heard It by Listener");

}

}

public class Subscriber2 //2nd subscriber class

{

public void Subscribe2(Publisher m) //get the object of pubisher class

{

m.Tick += HeardIt; //attach listener class method to publisher class delegate object

}

private void HeardIt(Publisher m, EventArgs e) //subscriber class method

{

System.Console.WriteLine("Heard It by Listener2");

}

}

class Test

{

static void Main()

{

Publisher m = new Publisher(); //create an object of publisher class which will later be passed on subscriber classes

Subscriber1 l = new Subscriber1(); //create object of 1st subscriber class

Subscriber2 l2 = new Subscriber2(); //create object of 2nd subscriber class

l.Subscribe(m); //we pass object of publisher class to access delegate of publisher class

l2.Subscribe2(m); //we pass object of publisher class to access delegate of publisher class

m.Start(); //starting point of publisher class

}

}

**Output:**

Heard It by Listener

Heard It by Listener2

Heard It by Listener

Heard It by Listener2

Heard It by Listener

.

.

.

(infinite times)

**Conventions**

The following important conventions are used with events:

* Event Handlers in the .NET Framework return void and take two parameters.
* The first parameter is the source of the event; that is the publishing object.
* The second parameter is an object derived from EventArgs.
* Events are properties of the class publishing the event.
* The keyword event controls how the event property is accessed by the subscribing classes.

**Example 2:**  *A publisher class* ***VideoEncoder*** *invokes the methods of 2 subscriber classes**(i.e. inform them)* ***MailService*** *and* ***MessageService*** *(when the video is encoded)**using delegate.*

**Program.cs:**

using System;

namespace EventsAndDelegates

{

class Program

{

static void Main(string[] arg)

{

VideoEncoder ve = new VideoEncoder(); //publisher

MailService mailsrc = new MailService(); //subscriber

MessageService msgsrc = new MessageService(); //subscriber

//we can further add more subscribers without changing the code of publisher!

//and publisher class will also invoke methods of newly created subscriber classes!!

mailsrc.Subscribe(ve); //pass the object of publisher class to MailSerive class

msgsrc.Subscribe(ve); //pass the object of publisher class to MessageService class

ve.Encode(); //entry point of videoEncoder class

}

}

}

**VideoEncoder.cs:**

using System;

using System.Threading;

namespace EventsAndDelegates

{

public class VideoEncoder

{

//STEPS:

//1. Define a delegate

public delegate void VideoEncodedEventHandler(object source, EventArgs args);

//2. Define an event based on that delegate ,event keyword is optional

public event VideoEncodedEventHandler VideoEncoded; //using past tense

public void Encode()

{

Console.WriteLine("Encoding video...");

Thread.Sleep(1000);

Console.WriteLine("video is encoded...");

Console.WriteLine("informing all subscriber classes...");

Thread.Sleep(1000);

OnVideoEncoded();

Console.ReadKey();

}

//3. Raise the event, good convention to use protected and virtual keyword

protected virtual void OnVideoEncoded() //using present tense

{

if (VideoEncoded != null)

VideoEncoded(this, EventArgs.Empty);

}

}

}

**MailService.cs**

using System;

namespace EventsAndDelegates

{

public class MailService

{

public void Subscribe(VideoEncoder ve)

{

ve.VideoEncoded += OnVideoEncoded;

}

public void OnVideoEncoded(object source, EventArgs e)

{

Console.WriteLine("MailService: Sending an email...");

}

}

}

**MessageService.cs:**

using System;

namespace EventsAndDelegates

{

public class MessageService

{

public void Subscribe(VideoEncoder ve)

{

ve.VideoEncoded += OnVideoEncoded;

}

public void OnVideoEncoded(object source, EventArgs args)

{

Console.WriteLine("MessageService: Sending a text message...");

}

}

}

**Output:**

Encoding video...

video is encoded...

informing all subscriber classes...

MailService: Sending an email...

MessageService: Sending a text message...