Tutorial – Delegates and Event Handling

In C#, events are a way for a class (a publisher) to provide notifications to clients (subscribers) when something interesting happened.

This is very common when we are working with graphical user interfaces – like Windows Forms. Forms contain controls, like buttons, and generally we want to know when interesting things happen to those controls (like, for example, when a button is clicked). Controls have events that are notified when these things happen, and we can customize how the program responds to these events by writing an event handler.

An event is a way for a class to allow clients to get it delegates to methods that should be called when the event occurs. Each time the event occurs, all the delegates specified for that event are invoked.

In this tutorial we will create a class containing a list. Whenever the list is changed, a Changed event will be envoked. We will then add some event handlers that will perform some processing in response to these change events.

This is a general-purpose example that could be used in numerous ways inside a larger program.

When making an AI Node Editing program we could, for example, store a list of nodes in our path. Whenever a node is added or removed from this list we could receive a notification of this event and perform some analysis of the nodes in our list – like making sure each node is reachable from every other node.

Create the Event Sender:

We'll override the standard ArrayList, and add functionality so that any time an element in the list is added or changed it will send an event to any clients attached via the delegate method.

1. Create a new C# Console Application
2. Add a new class called *ListWithChangedEvent*
3. Add the following code to the .cs file

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DelegateSample
{
    public delegate void ChangeEventHandler(object sender, EventArgs e);


    class ListWithChangedEvent: ArrayList
    {
        public ChangeEventHandler Changed;

        protected virtual void OnChanged(EventArgs e)
        {
            if (Changed != null)
                Changed(this, e);
        }

        public override int Add(object value)
        {
            int i = base.Add(value);
            OnChanged(EventArgs.Empty);
            return i;
        }

        public override void Clear()
        {
            base.Clear();
            OnChanged(EventArgs.Empty);
        }

        public override object this[int index]
        {
            set
            {
                base[index] = value;
                OnChanged(EventArgs.Empty);
            }
        }
    }
}
```

We start by declaring the delegate. Any function that we implement to receive messages from the *ListWithChangedEvent* class will need to have the same argument list.

The class itself stores an instance of the delegate. Delegate functions are added via the public *Add* function.

Create the Event Listener:

The event listener will be a very simple class that stores an instance of a *ListWithChangedEvent*, and contains the event handler – the function to be added to the List's delegate.

1. Create a new class called *EventListener*
2. Add the following code to the new class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DelegateSample
{
    class EventListener
    {
        private ListWithChangedEvent List;

        public EventListener(ListWithChangedEvent list)
        {
            List = list;
            List.Changed += new ChangeEventHandler(OnListChanged);
        }

        private void OnListChanged(object sender, EventArgs e)
        {
            Console.WriteLine("list changed event received");
        }

        public void Detatch()
        {
            List.Changed -= new ChangeEventHandler(OnListChanged);
            List = null;
        }
    }
}
```

Upon construction, the *EventListener* class attaches the *OnListChanged* function to the list's delegate. After this, the *OnListChanged* function will be called any time a change is made to the list (that is, any time an object is added to the list).

The Rest of the Program:

To get everything working we need an instance of the *ListWithChangedEvent*, and an instance of the *EventListener*. Modify the *Main()* function as follows:

```
static void Main(string[] args)
{
    // create a new list
    ListWithChangedEvent list = new ListWithChangedEvent();

    // creeat a class that listens for when the list is changed
    EventListener listener = new EventListener(list);

    list.Add("hello");
    list.Add("world");
    list.Clear();
    listener.Detatch();


    Console.ReadKey();
}
```

Execute the program and you should see the message "*list change event received*" displayed on the console three times.

Exercise:

Create a console program that contains a keyboard handling class that sends out events when keys are pressed or released.

Your program should have an update loop (game loop) that will update the keyboard class every iteration. The keyboard class can then check the current state of the keyboard and send the appropriate events to any delgates attached.

Hint:
Use the following code inside your keyboard Update function to query whether or not the console has had input

```
public void Update()
{
    if(Console.KeyAvailable == true)
    {
        cKeyInfo = Console.ReadKey();

        // send the key to the delegate
    }
}
```