Delegate Tutorial for Beginners

Amey K Bhatkar

28 Jul 2013 CPOL

Rate this: 4.48 (20 votes)

Delegate tutorial for beginners.

Download source code - 111.5 KB

Introduction

This is one of the articles from the series: "Delegate tutorial for beginners". This tutorial demonstrates how to create and use simple delegates and the different types.

Content

- What is Delegate type?
- What advantage does delegate makes over method invocation?
- What is a Multicast Delegate?
- What is the relation between Event and Delegate?
- What is an Asynchronous Delegate?

What is Delegate type?

Delegate is a reference type, it references to a method having the same signature as the delegate.

Creation of a delegate is a three step process:

1. Define the Delegate
2. Create an object of the Delegate and assign a method to it
3. Invoke the Delegate

1. Define the Delegate

Following is the way to define a delegate:

Hide Copy Code

```
delegate returntype delegateName (parameter);
```

[Return type and parameter define the signature of the Delegate.]

An object of the Delegate allows the programmer to encapsulate a reference to a method. We can assign any method to a Delegate object having the same signature as that of the delegate.

2. Create an Object of the Delegate and assign a method to it

We define the Delegate object as follows:

Hide Copy Code

```
delegateName objectOfDelegate=null;
```

We can assign a reference of the method to the delegate object as follows:

Hide Copy Code

```
objectOfDelegate=functionName;
```

Note: The signature of the function and Delegate must be the same.

3. Invoke the Delegate

You can invoke a Delegate by using the `Invoke` method of the Delegate:

Hide Copy Code

```
objectOfDelegate.Invoke(parameterValue);
```

Invocation of the Delegate is the same as the invocation of the method that is encapsulate in the Delegate.

Let's take a sample example. Consider a function that accepts two numbers and adds them.

Let's create a Delegate that points to an Add function.

1. Define the Delegate

Hide Copy Code

```
delegate void PointToAdd(int numberOne,int numberTwo);
```

2. Create an object of the Delegate and assign a method to it

Hide Copy Code

```
PointToAdd pointToAdd = AddTwoNumber;
```

3. Invoke a Delegate

Hide Copy Code

```
pointToAdd.Invoke(10, 20);
```

Now let's look at a complete sample at once.

Hide Copy Code

```
class DelegateDemo1
{
    delegate void PointToAdd(int numberOne,int numberTwo);
    static void Main(string[] args)
    {
        PointToAdd pointToAdd = AddTwoNumber;
        pointToAdd.Invoke(10, 20);
```

```
        Console.Read();
    }
    static void AddTwoNumber(int numberOne,int numberTwo)
    {
        Console.WriteLine((numberOne+numberTwo).ToString());
    }
}
```

Invocation of the Delegate is the same as the invocation of the method that is encapsulate by the Delegate. So is it true that we can replace the Delegate with a simple method invocation?

What advantage does a Delegate have over a method invocation?

The developer can assign a method to a Delegate during runtime. Do you remember pointers in C? A Delegate is nothing but a pointer to a function. When developers are not aware which function to call or a function call is based on a condition, they can choose a delegate to call the function during run time.

Let's take sample example. Similar to our above example we will create functions for adding, subtracting, division, and multiplication of two numbers.

We give the choice to the user to enter two numbers and a string, adds for addition, subs for subtraction, and div for division of two numbers.

Let's define the Delegate that supports a simple math operation of two numbers.

Hide Copy Code

```
public delegate void PointToMathsOperation(int numberOne, int numberTwo);
PointToMathsOperation pointToMathsOperation = null;
```

We have assigned the Delegate to `null`. We will assign a method to the Delegate dynamically or at run time. We have the following methods matching the delegate signature.

Hide Copy Code

```
private void Add(int numberOne, int numberTwo)
{
    Console.WriteLine((numberOne + numberTwo).ToString());
}

private void Sub(int numberOne, int numberTwo)
{
    Console.WriteLine((numberOne - numberTwo).ToString());
}

private void Div(int numberOne, int numberTwo)
{
    Console.WriteLine((numberOne / numberTwo).ToString());
}

private void Mul(int numberOne, int numberTwo)
{
    Console.WriteLine((numberOne * numberTwo).ToString());
}
```

Let's write a method that will assign a method to the Delegate based on the user choice and return the object of the Delegate.

Hide Copy Code

```
public PointToMathsOperation Operation(string operationName)
{
    switch (operationName)
    {
        case "Add":
            pointToMathsOperation = Add;
```

```
            return pointToMathsOperation;
        case "Sub":
            pointToMathsOperation = Sub;
            return pointToMathsOperation;
        case "Div":
            pointToMathsOperation =Mul;
            return pointToMathsOperation;
        case "Mul":
            pointToMathsOperation = Div;
            return pointToMathsOperation;
    }
    return pointToMathsOperation;
}
```

Let's take a compete look at what we have done so far. We have defined a `Math` class as follows:

Hide Shrink Copy Code

```
public class Math
{
    public delegate void PointToMathsOperation(int numberOne, int numberTwo);
    PointToMathsOperation pointToMathsOperation = null;
    public PointToMathsOperation Operation(string operationName)
    {

        switch (operationName)
        {
            case "Add":
                pointToMathsOperation = Add;
                return pointToMathsOperation;
            case "Sub":
                pointToMathsOperation = Sub;
                return pointToMathsOperation;
            case "Div":
                pointToMathsOperation =Mul;
                return pointToMathsOperation;
            case "Mul":
                pointToMathsOperation = Div;
                return pointToMathsOperation;
        }
        return pointToMathsOperation;
    }
    private void Add(int numberOne, int numberTwo)
    {
        Console.WriteLine((numberOne + numberTwo).ToString());
    }
    private void Sub(int numberOne, int numberTwo)
    {
        Console.WriteLine((numberOne - numberTwo).ToString());
    }
    private void Div(int numberOne, int numberTwo)
    {
        Console.WriteLine((numberOne / numberTwo).ToString());
    }
    private void Mul(int numberOne, int numberTwo)
    {
        Console.WriteLine((numberOne * numberTwo).ToString());
    }
}
```

Let's instantiate the `Math` class and call the method `Operation` that returns the Delegate object. We can invoke the Delegate by passing two numbers to it as per the delegate signature.

Hide Copy Code

```
public class DelegateDemo2
{
    public static void Main(string[] args)
    {
```

```
        Console.WriteLine("Please enter two number:-");
        var numberOneString = Console.ReadLine();
        var numberTwoString = Console.ReadLine();
        Console.WriteLine("Please enter operation to perform like add,mul,sub and div :-");
        var choice = Console.ReadLine();
        int numberOne, numberTwo;
        Math math = new Math();
        int.TryParse(numberOneString,out numberOne);
        int.TryParse(numberTwoString,out numberTwo);
        math.Operation(choice).Invoke(numberOne,numberTwo);
        Console.ReadLine();
    }
}
```

What is a Multicast Delegate?

Multicast Delegate is an extension of a Delegate. You can assign multiple methods to a Delegate object and invoke them during Delegate invocation. Methods get invoked in the same hierarchy in which they get assigned to the Delegate object.

Let's take a simple example. Consider a system that will broadcast messages when it fails doing transactions. The system will broadcast a message to the Admin via SMS on mobile, email on mailbox, and message on fax. For simplicity we will build a console application which accepts a string from the user, any string content other than "Pass" assumes that the transaction has failed, and we broadcast a message to the admin using a Multicast Delegate.

Let's build a `BroadCast` class that contains a method to transmit a message to the admin via SMS on mobile, email on mailbox, and message on fax.

Hide Copy Code

```
/// <summary>
/// broadcast done.
/// </summary>
public class BroadCast
{
    public void ViaMessage()
    {
        Console.WriteLine("Message Send to mobile.");
    }

    public void ViaEmail()
    {
        Console.WriteLine("Email send to mailbox.");
    }

    public void ViaFax()
    {
        Console.WriteLine("Message send via fax.");
    }
}
```

Our system can broadcast a message using the above media. Let's build a system that will broadcast a message when it fails using a Multicast Delegate.

Hide Shrink Copy Code

```
/// <summary>
/// Multicast Delegate
/// </summary>
public class DelegateDemo3
{
    delegate void SendMessage();

    public static void Main(string[] args)
    {
        BroadCast broadCast=new BroadCast();
        SendMessage sendMessage = broadCast.ViaEmail;
        sendMessage += broadCast.ViaFax;
```

```
        sendMessage += broadCast.ViaMessage;
        Console.WriteLine("Enter trascation suncess.");
        var result = Console.ReadLine();
        if (result.Equals("Pass"))
        {
            Console.WriteLine("Successfull trasaction!");
        }
        else
        {
            Console.WriteLine("Transaction fail !");
            sendMessage.Invoke();
        }
        Console.ReadLine();
    }
}
```

What is the relation between an Event and a Delegate?

Consider a publisher who published something and there are subscribers who wait for the publisher to publish so they can start their work. There is actually no dependency between the subscriber and publisher. The subscriber will do its work independent of the publisher, they are independent agents but the subscriber needs an alert from the publisher to start its activity.

Let us relate this with our Multicast Delegate where our delegate is the publisher. It publishes the event broadcast message where our subscriber has complete functionality to broadcast a message on fax, cell, and email but they only want to perform their activity when the publisher publishes the broadcast event.

So we have our multicast delegate as our publisher and subscriber model.

Good enough but as we have stated earlier, delegates are pointers to functions, i.e., they reference to a function. We can set a reference as null, i.e., we can set a pointer as null also.

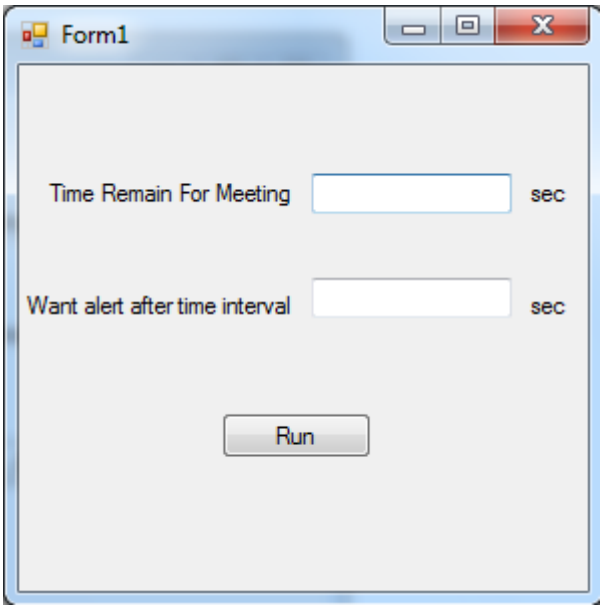
In our case the subscriber can null the publisher. The Subscriber can attach a method to the publisher object.

But we have said that the subscriber and publisher are independent agents, i.e., there is a decoupling between the subscriber and publisher. So we are breaking the rule of the subscriber publisher model. In order to maintain this model we can use the event which sits on top of the Delegate and encapsulates the Delegate.

Events are nothing but alerts to an action. If we performing an action of button click then we are raising the click event that just tells that button click is done. It does not enforce any rule on what needs to be done when the button is clicked. The event is an alert to the external stimuli or action on an object in object oriented programming. The subscriber can only listen to an event.

Consider a scenario where a group of people are subscribers interested to attend the meeting after a predefined time. They want an alert from the publisher after each predefined interval of time so they can sync their own activity accordingly.

Let's build a UI that will accept two numbers. The first number indicates the time remaining for a meeting and the second indicating the alert wanted after the time interval periods.



If we select time remaining for meeting as 12 and want alert after time interval of 2, then we get an alert every 2 seconds for the meeting so will get a total of five alerts.

Let's start building the above requirement. We will start with defining our Delegate.

Hide Copy Code

```
public delegate void TimeReachedEventHandler(object sender, TimeReachedEventArgs e);
```

This delegate will encapsulate any method that returns `void` and takes two parameters. The first parameter is an object that represents the sender (the object raising the event), and the second parameter is an object of type `TimeReachedEventArgs`, derived from `EventArgs`, that will contain useful information for anyone interested in this event.

`TimeReachedEventArgs` is defined as follows:

Hide Copy Code

```
public class TimeReachedEventArgs : EventArgs
{
    private int _reached;
    public TimeReachedEventArgs(int num)
    {
        this._reached = num;
    }
    public int ReachedNumber
    {
        get
        {
            return _reached;
        }
    }
}
```

`TimeReachedEventArgs` has the information about the time interval at which subscriber wants an alert from the publisher. Let's build a class that contains the business logic for the event that will fire on every time interval elapsed till the meeting starts.

Hide Copy Code

```
public  event TimeReachedEventHandler TimeReached;
```

We have created our event but if you look at the code carefully you will find that our event is of type delegate. Apart from the event, we have the methods `AltertMe` and `OnNumberReached`. The `AlterMe` method accepts the total time for the meeting and the alert wanted after the time period. It starts a counter till it equals to the interval for the alert. Then it passed the time elapsed to `TimeReachedEventArgs`. Then it passed the `TimeReachedEventArgs` object to the `OnNumberReached` method.

Hide Copy Code

```
public void AltertMe(int timeToMeetingTime, int alertAfterTime)
{
    if (timeToMeetingTime < alertAfterTime)
        throw new ArgumentException("Alter time is less than the meeting time.");
    Stopwatch st = new Stopwatch();
    st.Start();
    while(true)
    {
        int interval=TimeSpan.FromTicks(st.ElapsedTicks).Seconds;
        if (interval == alertAfterTime)
        {
            TimeReachedEventArgs e = new TimeReachedEventArgs(alertAfterTime);
            OnNumberReached(e);
            st.Stop();
            return;//do not count any more;
        }
    }
}
```

The `OnNumberReached` method accepts the `TimeReachedEventArgs` object and invokes the event `TimeReached`.

Hide Copy Code

```
protected virtual void OnNumberReached(TimeReachedEventArgs e)
{
    TimeReached(this, e);
}
```

We have the following code in the back end of the UI that will listen to the event. On button click we have call the event for each time interval.

Hide Shrink Copy Code

```
public partial class Form1 : Form
{
    Counter _meetingTimer;
    public Form1()
    {
        InitializeComponent();
        _meetingTimer = new Counter();
        _meetingTimer.TimeReached += new TimeReachedEventHandler(AlterMe);
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            int timeToMeeting = Convert.ToInt16(txtTimeToMeeting.Text);
            int alertTime = Convert.ToInt16(txtAlertTime.Text);
            while (timeToMeeting > alertTime)
            {
                _meetingTimer.AltertMe(timeToMeeting, alertTime);
                alertTime += alertTime;
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }

    private void AlterMe(object s, TimeReachedEventArgs r)
    {
        MessageBox.Show("Alterting you after:- "+r.ReachedNumber);
    }
}
```

What is an Asynchronous Delegate?

Sometimes we need to perform a large task and we do not want to wait till the task ends, instead we prefer to move to the next task and when the large task is finished we want a notification that the large task has completed successfully or not. A real life situation and the answer is pretty simple Asynchronous execution of task.

Let's see what asynchronous execution is with the help of the following diagram.

Consider the above diagram. We have a large task that takes 15 minutes to execute and a small task that takes 2 minutes to execute.

1) We are processing synchronously and we allow the lager task to process first.

Our small task needs to wait till our large task gets finished. 15 minutes is the minimum wait time for task 2. As both the tasks are independent, it is not logical for our small task to wait till the large task finishes its execution.

2) Let's process our large task asynchronously.

We start processing our large task first but instead of waiting for the large task to be finished we move in parallel to task 2 or subsequent tasks in the pipeline and execute them; when our large task finishes its execution we will take output from it.

Let's build the above scenario with the help of a Delegate.

Hide Copy Code

```
public class DelegateDemo4
{
    delegate Guid DoComplexLogic();
    public static void Main(string[] args)
    {
        DoComplexLogic docomplexLogic = MyComplexLogic;
        docomplexLogic.BeginInvoke(new AsyncCallback(CallBackMyComplexTask), docomplexLogic);
        Console.WriteLine("New task is started!");
        Console.Read();
    }
    public static void CallBackMyComplexTask(IAsyncResult asyncResult)
    {
        DoComplexLogic doComplexLogic = (DoComplexLogic)asyncResult.AsyncState;
        Guid newGuid= doComplexLogic.EndInvoke(asyncResult);
        Console.WriteLine(newGuid);
    }
    public static Guid MyComplexLogic()
    {
        Thread.Sleep(3000);
        return Guid.NewGuid();
    }
}
```

When the developer call the Delegate using the `.BeginInvoke()` method, the developer is specifying asynchronous execution of the method that is encapsulated by the Delegate. When we use `.BeginInvoke()`, we need to use a callback mechanism, i.e., when our huge task has finished execution it will call the callback method.

We have captured the async result, i.e., when our huge task has finished execution it will send the result to the callback method. We use the `.EndInvoke()` method and pass the result of the execution to it to get the return value form the method execution.

We have finished our simple example of Asynchronous Delegates.

License

Share

About the Author

Amey K Bhatkar

Web Developer Lionbridge
India

Amey K Bhatkar, a "Microsoft .Net" Web Developer.
I am programmer by will and profession.
I have completed my MCA in 2011 and join software industry.
Presently I am working with Lion Bridge Technologies in Mumbai - India

Comments and Discussions

You must Sign In to use this message board.

Search Comments

First Prev Next

error on download
RRRCP 4-Jun-15 7:04

Can you please add a code file
Member 11093343 22-Jan-15 23:14

My vote of 1
Member 11236897 15-Nov-14 14:23

My vote of 3
geekyomega 7-Sep-14 7:33

The code block is wrong
itish 21-May-14 22:04

My vote of 4
Nitij 29-Jul-13 23:44

My vote of 5
Vitor Garcia 29-Jul-13 4:41

Re: My vote of 5
Amey K Bhatkar 29-Jul-13 4:46

Re: My vote of 5
Smitha Nishant 26-Aug-13 6:11

Re: My vote of 5
Amey K Bhatkar 23-Sep-13 6:58

Issues
DaveAuld 28-Jul-13 23:45

Re: Issues
Amey K Bhatkar 29-Jul-13 2:16

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink Layout: fixed | fluid