Delegates are always used with events and when a method is dynamically invoked.

Custom delegates are defined using the ‘delegate’ keyword. Behind the scenes .NET will generate a class that inherits from some other .NET framework delegate classes.

public delegate void TheDelegate(int theArgs);

similar to a method signature but using the delegate keyword. The delegate definition defines the valid return types and args that can be passed from the event raiser to the event handler. The delegate is kind of like a pipeline verifier and any handler must match the return and args.

Define a matching handler:

Public void TheHandler(int theArgs){….}

.NET has an abstract base class Delegate. It has 2 important properties called ‘Method’ and ‘Target’. Method is the method name and Target is the object instance the method resides in.

Delegate class also has a ‘GetInvocationList()’ method it inherits from MulticastDelegate. It returns a collection of Delegate[]. When an event is raised the Delegate list is executed sequentially.

The delegate keyword is a compiler trick to use the Delegate class as you can’t inherit from it directly. The delegate keyword will auto-generate a class that inherits from these classes.

To attach an event handler to a delegate we new-up the delegate;

TheDelegate del1 = new TheDelegate(TheHandler);

del1(10); //invokes the delegate

To add delegates to the invocation list:

TheDelegate del2 = new TheDelegate(TheHandler);

del1 += del2;

or:

del1 += new TheDelegate(TheHandler);

The word ‘Handler’ is typically used when defining delegates.

ILSpy – an app that can trace code back to the intermediate language or right back to the C#.

ilspy.net. If you select the il(intermediate language) option you can see the actual auto generated code.

Delegates can have a return type but in the case of more than 1 in an invocation list, the last delegate called wins.

Events

Defining an event involves the keyword ‘event’:

Public event TheDelegate event1;

Events are the standard way in the .NET framework to provide notifications.

MethodImplOptions.Synchronised – added to the top of a method and will synchronise multiple accesses to the method.

Delegate.Combine – adds a delegate to the invocation list. Same as += to the delegate instance.

Delegate.Remove – removes a delegate from the invocation list.

EventHandler – a delegate with no customisation. It’s a built in delegate we can use for notification only.

Events are raised by invoking the event like a method. The other option is to cast the event to it’s delegate type and invoking it.

It’s important to check before invoking an event to see if it’s null. If it is an exception will be raised.

BUT BEST PRACTICE is not to invoke the event directly like this. The best practice is to take the name of the event and create method **OnEventName();** which will invoke the event. This syntax is commonly used in .NET framework. Usually thr null check is done within the On… method.

When invoking an EventHandler delegate, for the sender param you can use ‘this’ and for EventArgs you can use EventArgs.Empty

When working with Events in .NET the data needs to be passed in a specific way. The standard is to use EventArgs and to send a sender object.

When customised data needs to be sent we extend the EventArgs class. By doing this we don’t need to keep adding params to delegate definitions which keeps the signatures of the delegate and the event and any using methods clean. All the data is encapsulated by EventArgs.

We should always send data in a class that inherits from EventArgs:

public class CustomEventArgs : System.EventArgs

{

public int Time{ get; set;}

}

CLR object. POCO class…..? A simple object created in the Common Language Runtime of the .NET framework. **P**lain **O**ld **C**LR **O**bject.

.NET provides a generic EventHandler<T> that allos use to pass custom data without needing to define our own delegate with custom data:

public event EventHandler<CustomEventArgs> WorkPerformed;

This method is preferred as the compiler can generate the delegate.