

.NET FUNDAMENTALS: EVENTS

Bill Sheldon

- Microsoft MVP - Visual Basic
- Senior Technical Analyst-Developer Rubio's Restaurants
- Author Office Business Application Development (WROX)
- Instructor UCSD Extension
- Author Professional Visual Basic (WROX)
- Author SQL Server Magazine

Events

- ▣ Events are at the core of the Windows OS
 - At the core of any interface – most of us familiar with use in Windows Forms
 - Events refer to the ability to tell another piece of code that something has happened
- ▣ There are two parts to events in .NET
 - Consuming events, in particular those produced by the system
 - Raising events, in particular those you define
- ▣ Events are a sub class of Delegates... do we need to discuss delegates?

Events

- ▣ Traditional application (no Events)
 - Application starts and executes commands
 - User may be prompted on command line for input or inputs may be provided at startup
 - Application may include a loop to allow user repeated actions within the flow
 - Application completes
- ▣ Events make take the loop concept and abstract it from the application
 - Application can now support input from user or OS and respond dynamically to both

Events

- ▣ An event can take place due to action taken by:
 - The user of an application
 - The application code
 - The operating system
- ▣ When an event occurs it is said to have been *triggered*, *raised* or *fired*
- ▣ Events generally have arguments that identify specific data that pertain to the event fired
- ▣ Each event has a source, said to be the sender
- ▣ An event handler is a procedure that is executed as a result of event notification
 - The process of declaring an event handler for an event is called binding a procedure to an event

Events

- ▣ The operating system and other event based system have at their core a message pump/queue
 - All events go to this module and then applications are notified of the next message of interest
- ▣ Each application, although often single threaded contains a message loop
 - While executing the handler for an event the loop is occupied.
 - Once event handling is complete the loop executes again
 - When an app is running; it is looking for events

Events

- ▣ Coupling and Cohesion
 - If you aren't familiar with these concepts they are one way to look for good software design
 - Coupling indicates how dependent different components/methods are on knowing each other.
 - Cohesion references how many different things a given component/method tries to do in a single call.

Events

- ▣ We want High cohesion and Low coupling
 - Events support this model almost perfectly
 - An event represents a single action (High cohesion)
 - An event may be handled by any one or more registered components/methods – without the triggering class being aware of who/what is handling the event (Low coupling)
- ▣ Any application or class can register for an event without the owner being involved
- ▣ Make a great way for User Controls to notify the Application Frame and other User Controls of status

Events

- ▣ ASP.NET has a 'Message Pump'
 - It's called IIS
 - When a ASP.NET form action occurs, an 'event' is sent back to IIS
 - The application then responds to this 'event' with HTML that is returned to the client
- ▣ Although the implementation is completely different and doesn't use the Event class the underlying concept is the same.

Handling Events

Looking at handling events that have been raised

Handling Events

- ▣ This comes naturally in .NET for most UI developers you just double click...
 - OK, in reality when you do that Visual Studio is generating a good bit of code for you, but that's not how a new developer sees it.
- ▣ In reality handling most Windows.Forms and Windows events is automated by Visual Studio
 - But professional developers go beyond what Visual Studio generates
 - Need to be able to handle any event

Handling Events

- ▣ C# references the event property and adds or subtracts the handler
- ▣ To add an event handler:

```
Button1.Click += Button1_Click;
```
- ▣ To remove an event handler:

```
Button1.Click -= Button1_Click;
```
- ▣ Note that Button1 provides an interface to allow adding handlers – but doesn't know at compile time who will handle the events.
 - Handler assignments in .NET are dynamic

Handling Events

- ▣ In VB there are two methods, one acts static and will be covered later.
- ▣ The AddHandler and RemoveHandler keywords manage events dynamically
- ▣ To Add an event handler
`AddHandler (button1.Click, AddressOf(Button1_Click))`
- ▣ To remove an event handler
`RemoveHandler(button1.Click, button1.Click, AddressOf(Button1_Click))`

Handling Events

- ▣ Keep in mind that an event handler is an object reference
- ▣ Thus if Form1 adds a handler call to Form2, Form2 now has a reference to Form1
- ▣ If Form1 is no longer needed, the garbage collector won't clean it up as long as Form2 references it for a handler.
- ▣ Always mirror any added handlers with removal of those handlers in your dispose method.

Delegates

Events like threading spawn from Delegates

Delegates

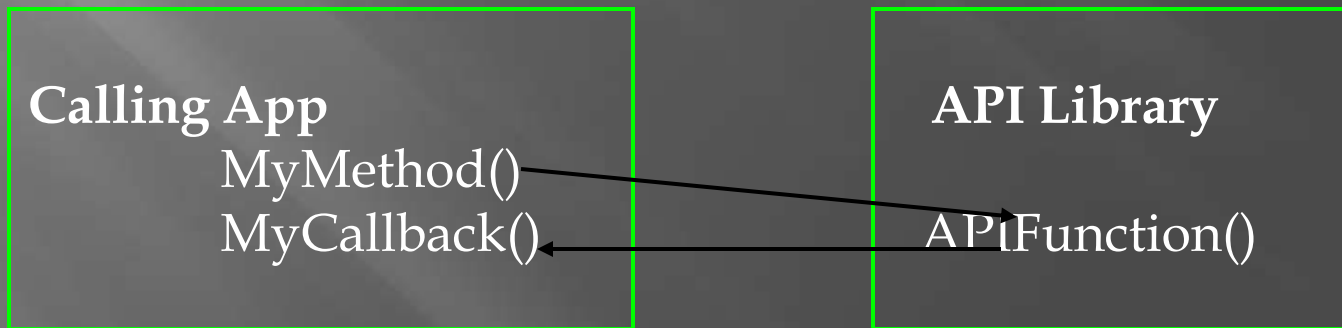
- ▣ In some low-level languages (like C and C++) there is a variable type known as a pointer
 - Pointers are values that "point" to a specific memory address, similar to reference-type variables
 - Variables and functions can exist at the address pointed to by a pointer
- ▣ .NET developers are generally "protected" from using pointers, as their misuse can lead to disastrous results ranging from program termination to data corruption
- ▣ However, the need to create references to functions has outweighed the need for deliberate protection
- ▣ Delegates are a "safe" alternative to the use of function pointers

Delegates

- ▣ Delegate Methods and Properties:
 - Target –the object instance owning the method to be called
 - Method – A MethodInfo object that describes the method(s) associated with the target
 - 'Invoke' –<implicit> Invokes a target method(s)
- ▣ To create a new delegate
 - Delegate(Object, String) – creates an instance call
 - Delegate(Type, String) – creates a call to a static method
- ▣ While the delegate has internal properties typically you will use the constructor instead of the Target and Method properties.

The Delegate as a Callback

- ▣ Many Windows API functions require the use of a callback function
 - Some functions need to communicate with their calling routine
 - When the called function needs to talk back to the calling routine, it will execute the given callback method
 - Similar to events, callback execution can happen at any time



Using a Delegate to Call a Method

- ▣ Delegates can also be used to call a method not in an external API
- ▣ This process is useful if you want to create a method definition (one with a specific set of parameters) and call different methods with the same definition
- ▣ Events make heavy use of delegates and their function will become clearer with examples

Creating Events

How to create your own custom event implementations and raise events

Custom Events

- ▣ In addition to the events defined for controls and Window Forms (and other objects), .NET supports the creation of custom events
- ▣ Custom events are useful in classes to indicate when certain actions occur
 - Update a status in main window when value in control changes
 - Trigger validation when user moves from one user control to another
 - Support dynamic loading of objects where the status of each object needs to be checked when changed

Custom Events

- ▣ VB supports a short cut syntax for custom events
Events are declared very similarly to delegates using the Event keyword
`Public Event <Name> (<Arguments>)`
- ▣ Don't need to explicitly define the delegate
- ▣ Works the same as the C# custom event logic just VB creates the event in a more declarative manner... but not very efficient
- ▣ An explicit handler can be more efficient

Custom Events

- ▣ The disadvantage of the VB method includes
 - Multiple similar delegates that use a common handler could be generated – ie. Potentially inefficient
 - Non-VB developers get lost looking for plumbing
- ▣ The preferred method is to declare a delegate as the handler

```
public delegate void  
    <event>EventHandler(arguments);
```

```
Public Delegate Sub  
    <event>EventHandler(arguments)
```

Custom Events

- ▣ Sidebar...
 - Events support passing arguments
 - Event handlers must have a number of arguments that match the arguments defined for an event.
 - For a simple 'String' or 'Int' value there is nothing wrong with using this as the argument
- ▣ Best practice is to create an `<event>_EventArgs` class to hold multiple values
 - Replace (arguments) with your class
(MyCustomEventArgs e)

Custom Events

- ▣ Once you have the delegate, associate it with a public event in your class

```
public event <event>EventHandler  
    <event>;
```

```
Public Event <event> As  
    <event>EventHandler
```

- ▣ Yes VB and C# use reverse order...
- ▣ That's it you have an event ready to be consumed like any other..

Custom Events

- ▣ Raising an event – the easy part

- ▣ Syntax for VB

`RaiseEvent <event>(arguments)`

- ▣ Syntax for C#

`<event>(arguments);`

- ▣ This triggers the event now you need to handle it in your code... just like a windows event as described earlier

Declarative Events

Looking at how things like XAML and VB use declarative logic to define event handlers

Declarative Event Handlers

- ▣ XAML is a declarative language by design
 - This is ‘what’ I want; ‘how’ it happens is hidden
`<button click=“button1_click”>`
 - Declares an event handler that needs to appear somewhere in the backing code
- ▣ Don’t need the `AddHandler` or `class.Event +=` dynamic call in the backing code
- ▣ XAML will look for a method matching the one declared and assign the handler automatically
 - One drawback – you have now coupled your XAML to a named method in your backing code

Declarative Event Handlers

- ▣ VB provides certain constructs which are declarative in nature – “Handles”
 - In VB the WithEvents keyword is used when declaring the object to support the Handles clause
- ▣ Allows you to declare that a given method ‘handles’ the event

```
Public Sub Button1_Click() Handles Button1.Click
```
- ▣ Visual Studio uses this syntax
 - Also introduces limited coupling but in this case to an object that may be defined in XAML... XAML artist is unconcerned with handlers

Conclusion

- ▣ Events are a great tool for building complex interactions
- ▣ Provide a low coupling interface with high cohesion
- ▣ Visual Studio generates common event handlers
- ▣ You can and should create your own custom events to communicate between your components
- ▣ Best practice when creating events is to use an EventArgs object if you have multiple values to pass

Questions

- ▣ My slides are posted on my blog:
<http://www.nerdnotes.net/blog>