Chapter 35: Events

Section 35.1: Sources and Handlers

What are events?

VBA is *event-driven*: VBA code runs in response to events raised by the host application or the host document - understanding events is fundamental to understanding VBA.

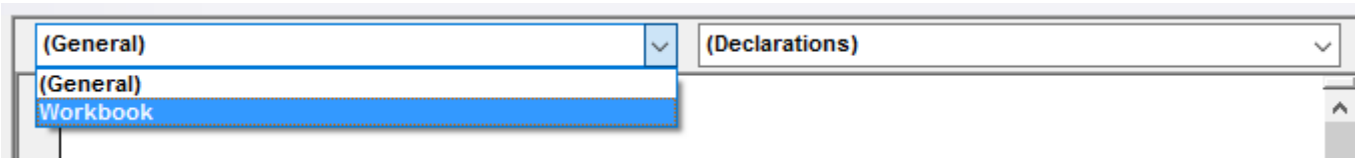
APIs often expose objects that raise a number of *events* in response to various states. For example an `Excel.Application` object raises an event whenever a new workbook is created, opened, activated, or closed. Or whenever a worksheet gets calculated. Or just before a file is saved. Or immediately after. A button on a form raises a `Click` event when the user clicks it, the user form itself raises an event just after it's activated, and another just before it's closed.

From an API perspective, events are *extension points*: the client code can chose to implement code that *handles* these events, and execute custom code whenever these events are fired: that's how you can execute your custom code automatically every time the selection changes on any worksheet - by handling the event that gets fired when the selection changes on any worksheet.

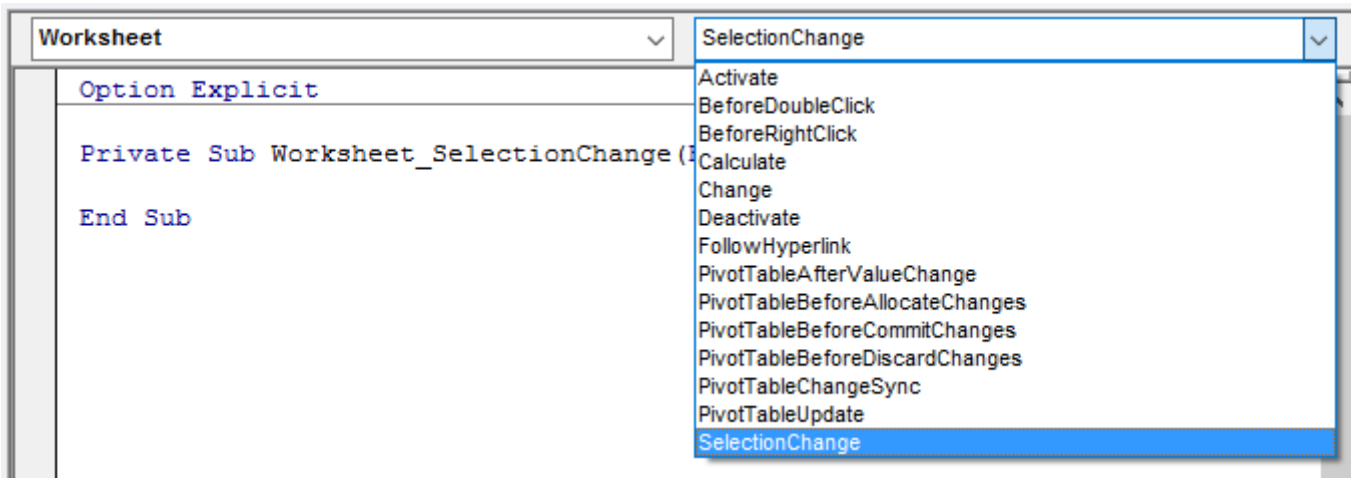
An object that exposes events is an *event source*. A method that handles an event is a *handler*.

Handlers

VBA document modules (e.g. `ThisDocument`, `ThisWorkbook`, `Sheet1`, etc.) and `UserForm` modules are *class modules* that *implement* special interfaces that expose a number of *events*. You can browse these interfaces in the left-side dropdown at the top of the code pane:



The right-side dropdown lists the members of the interface selected in the left-side dropdown:



The VBE automatically generates an event handler stub when an item is selected on the right-side list, or navigates there if the handler exists.

You can define a module-scoped `WithEvents` variable in any module:

```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet
```

Each `WithEvents` declaration becomes available to select from the left-side dropdown. When an event is selected in the right-side dropdown, the VBE generates an event handler stub named after the `WithEvents` object and the name of the event, joined with an underscore:

```
Private WithEvents Foo As Workbook
Private WithEvents Bar As Worksheet

Private Sub Foo_Open()

End Sub

Private Sub Bar_SelectionChange(ByVal Target As Range)

End Sub
```

Only types that expose at least one event can be used with `WithEvents`, and `WithEvents` declarations cannot be assigned a reference on-the-spot with the `New` keyword. This code is illegal:

```
Private WithEvents Foo As New Workbook 'illegal
```

The object reference must be `Set` explicitly; in a class module, a good place to do that is often in the `Class_Initialize` handler, because then the class handles that object's events for as long as its instance exists.

Sources

Any class module (or document module, or user form) can be an event source. Use the `Event` keyword to define the *signature* for the event, in the *declarations section* of the module:

```
Public Event SomethingHappened(ByVal something As String)
```

The signature of the event determines how the event is raised, and what the event handlers will look like.

Events can only be *raised* within the class they're defined in - client code can only *handle* them. Events are raised with the `RaiseEvent` keyword; the event's arguments are provided at that point:

```
Public Sub DoSomething()
    RaiseEvent SomethingHappened("hello")
End Sub
```

Without code that handles the `SomethingHappened` event, running the `DoSomething` procedure will still raise the event, but nothing will happen. Assuming the event source is the above code in a class named `Something`, this code in `ThisWorkbook` would show a message box saying "hello" whenever `test.DoSomething` gets called:

```
Private WithEvents test As Something

Private Sub Workbook_Open()
    Set test = New Something
    test.DoSomething
End Sub

Private Sub test_SomethingHappened(ByVal bar As String)
'this procedure runs whenever 'test' raises the 'SomethingHappened' event
```

```
    MsgBox bar
End Sub
```

Section 35.2: Passing data back to the event source

Using parameters passed by reference

An event may define a `ByRef` parameter meant to be returned to the caller:

```
Public Event BeforeSomething(ByRef cancel As Boolean)
Public Event AfterSomething()

Public Sub DoSomething()
    Dim cancel As Boolean
    RaiseEvent BeforeSomething(cancel)
    If cancel Then Exit Sub

    'todo: actually do something

    RaiseEvent AfterSomething
End Sub
```

If the `BeforeSomething` event has a handler that sets its `cancel` parameter to `True`, then when execution returns from the handler, `cancel` will be `True` and `AfterSomething` will never be raised.

```
Private WithEvents foo As Something

Private Sub foo_BeforeSomething(ByRef cancel As Boolean)
    cancel = MsgBox("Cancel?", vbYesNo) = vbYes
End Sub

Private Sub foo_AfterSomething()
    MsgBox "Didn't cancel!"
End Sub
```

Assuming the `foo` object reference is assigned somewhere, when `foo.DoSomething` runs, a message box prompts whether to cancel, and a second message box says "didn't cancel" only when [No] was selected.

Using mutable objects

You could also pass a copy of a mutable object `ByVal`, and let handlers modify that object's properties; the caller can then read the modified property values and act accordingly.

```
'class module ReturnBoolean
Option Explicit
Private encapsulated As Boolean

Public Property Get ReturnValue() As Boolean
'Attribute ReturnValue.VB_UserMemId = 0
    ReturnValue = encapsulated
End Property

Public Property Let ReturnValue(ByVal value As Boolean)
    encapsulated = value
End Property
```

Combined with the `Variant` type, this can be used to create rather non-obvious ways to return a value to the caller:

```
Public Event SomeEvent(ByVal foo As Variant)

Public Sub DoSomething()
    Dim result As ReturnBoolean
    result = New ReturnBoolean

    RaiseEvent SomeEvent(result)

    If result Then ' If result.ReturnValue Then
        'handler changed the value to True
    Else
        'handler didn't modify the value
    End If
End Sub
```

The handler would look like this:

```
Private Sub source_SomeEvent(ByVal foo As Variant) 'foo is actually a ReturnBoolean object
    foo = True 'True is actually assigned to foo.ReturnValue, the class' default member
End Sub
```