The MaxGUI Beginner Tutorial Series - Tutorial 15: EventHooks

(c) Assari Jan 13 2005

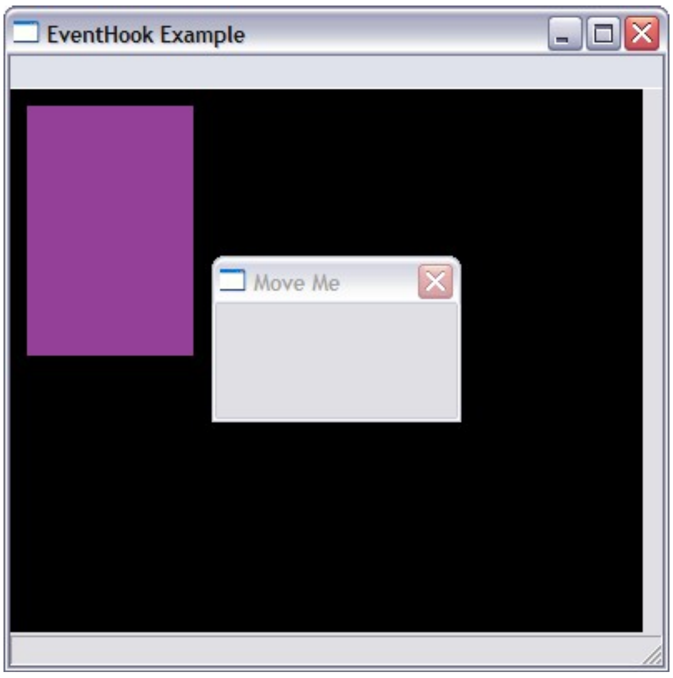
I was not thinking of covering this topic until much later as I don't quite consider eventhooks beginner stuff. But there has been some requests for this so I did a bit of research and now think that a "beginner" type tutorial on this subject is possible. So here we go...

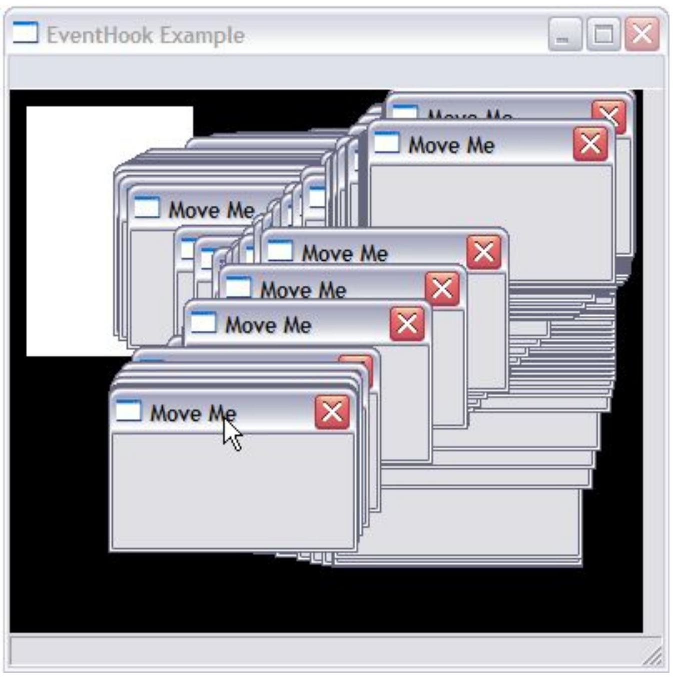
The Blitzmax event system uses the **WaitEvent**() function to wait for system events to occur. **WaitEvent**() however, has a limitation in that it will only allow you to process events once the user action such as releasing the mouse button has occured.

Let us take a look at a simple example to make this clear

```
SuperStrict

Local x:Int=(GadgetWidth(Desktop())-400)/2
Local y:Int=(GadgetHeight(Desktop())-400)/2

Local MyWindow:TGadget=CreateWindow("EventHook Example", x,y,400,400)
Global MyCanvas:TGadget=CreateCanvas(0,0,380,360,MyWindow)

Local SmallWindow:TGadget=CreateWindow("Move Me", x+125, y+150, 150, 100,
MyWindow, WINDOW_TITLEBAR)

Repeat
  WaitEvent()
  Select EventID()
  Case EVENT_WINDOWCLOSE
     End
  Case EVENT_GADGETPAINT
    UpDateCanvas()
   End Select
Forever
End

Function UpdateCanvas:Int()
     SetGraphics CanvasGraphics(MyCanvas)
     Cls
     SetColor Rnd(255),Rnd(255),Rnd(255)
     DrawRect 10,10,100,150
     Flip
End Function
```

If you were to then move this little window (drag by it's title bar), you will see a trail of images which are not deleted until you release the left mouse button. This is due to the WaitEvent function not receiving any event until the left button is released.





Before we proceed, let us make sure we understand the program we have just executed.

With the formula below, the 400x400 window created with the **x** and **y** coordinates will be centred on our desktop.

```
Local x:Int=(GadgetWidth(Desktop())-400)/2
Local y:Int=(GadgetHeight(Desktop())-400)/2
```

We then create our window, canvas and the small window which will appear on top of our canvas.

```
Local MyWindow:TGadget=CreateWindow("EventHook Example", x,y,400,400)
Global MyCanvas:TGadget=CreateCanvas(0,0,380,360,MyWindow)

Local SmallWindow:TGadget=CreateWindow("Move Me", x+125, y+150, 150, 100,
MyWindow, WINDOW_TITLEBAR)
```

There is our usual loop to either wait to end the program or repaint our canvas depending on the id of the event that we receive from **WaitEvent**()

```
Repeat
  WaitEvent()
  Select EventID()
  Case EVENT_WINDOWCLOSE
     End
  Case EVENT_GADGETPAINT
    UpDateCanvas()
   End Select
Forever
End
```

When we update our canvas, we draw a 100x150 rectangle of a random color.
(I could have just done a clearscreen, but I thought by drawing different colored rectangles we can see something happening)

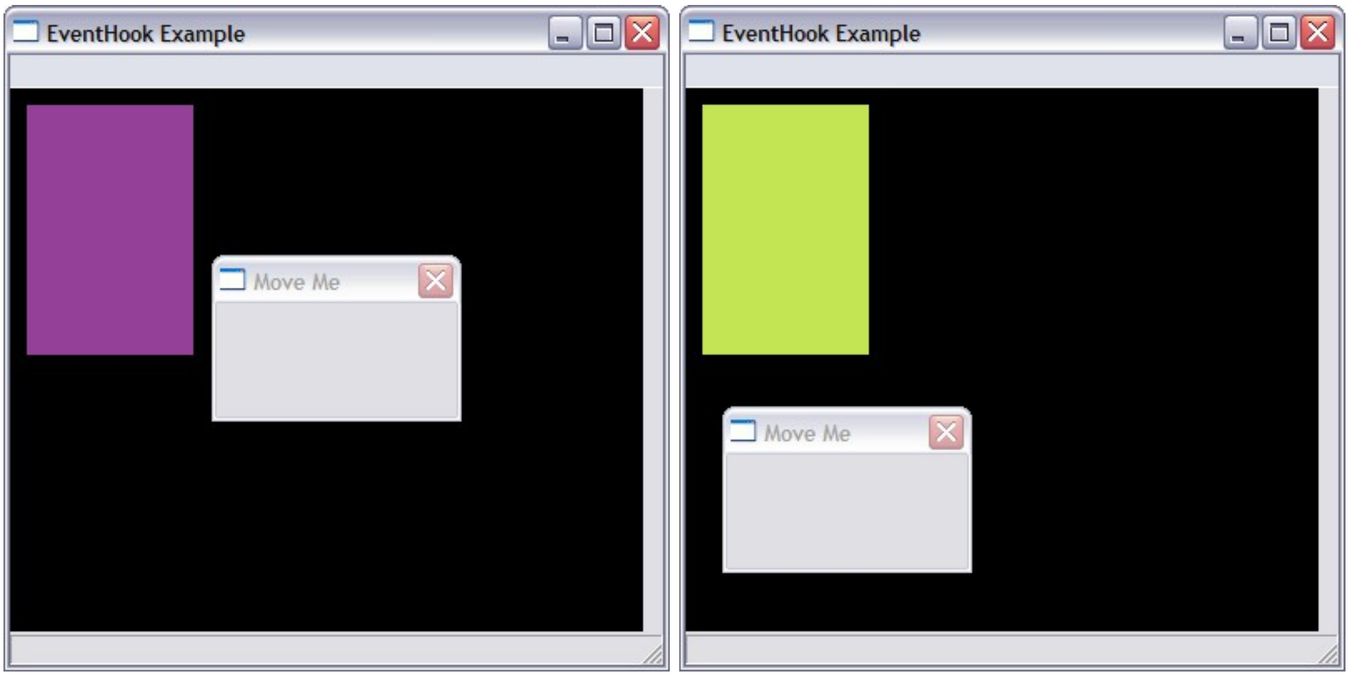
```
Function UpdateCanvas:Int()
     SetGraphics CanvasGraphics(MyCanvas)
     Cls
     SetColor Rnd(255),Rnd(255),Rnd(255)
     DrawRect 10,10,100,150
     Flip
End Function
```

EVENTHOOK

Obviously having a trail when we drag our window is not really acceptable. Now this is where the magic of eventhook comes in. Let us modify the above program and add in an eventhook (Note the addition in bold).

```
SuperStrict

Local x:Int=(GadgetWidth(Desktop())-400)/2
Local y:Int=(GadgetHeight(Desktop())-400)/2

Local MyWindow:TGadget=CreateWindow("EventHook Example", x,y,400,400)
Global MyCanvas:TGadget=CreateCanvas(0,0,380,360,MyWindow)

Local SmallWindow:TGadget=CreateWindow("Move Me", x+125, y+150, 150, 100,
MyWindow, WINDOW_TITLEBAR)

AddHook EmitEventHook, MyHook

Repeat
  WaitEvent()
  Select EventID()
  Case EVENT_WINDOWCLOSE
     End
  Case EVENT_GADGETPAINT
    UpDateCanvas()
   End Select
Forever
End

Function MyHook:Object(iId:Int,tData:Object,tContext:Object)
  Local Event:TEvent=TEvent(tData)

  If Event.source=MyCanvas And Event.ID=EVENT_GADGETPAINT
     UpdateCanvas()
     Return Null
  EndIf

  Return tData
End Function

Function UpdateCanvas:Int()
     SetGraphics CanvasGraphics(MyCanvas)
     Cls
     SetColor Rnd(255),Rnd(255),Rnd(255)
     DrawRect 10,10,100,150
     Flip
End Function
```

As we can now see, our little window is no longer leaving a trail of itself on the canvas. Our eventhook function has done its magic.



Let us try to understand what has happened. Let us take a look at the additions we have made to our original program.

This one liner below adds the hook into the event system to allow us to "hijack" an event when it happens way deep in the system internals.

```
AddHook EmitEventHook, MyHook
```

The syntax for **AddHook** function from the manual is as below:-

```
Function AddHook( id,func:Object( id,data:Object,context:Object
),context:Object=Null,priority=0 )
```

For our purposes we can actually simplify the syntax to:-

```
Function AddHook( id,func)
```

With id = **EmitEventHook** and **func** the name of the hook function which will be executed when the event is hijacked.

The Hook Function

Now lets take a look at the hook function itself.

There is a requirement that the hook function must always be defined as follows:-

```
Function MyHook:Object(iId:Int,tData:Object,tContext:Object)
```

With the only leeway given is to change the name of the function **MyHook** to basically anything you like (subject to Blitzmax's constraints on naming of course) as well as what we call the variables.

The following line casts the tData variable (which is a generic object) into an event object.

```
Local Event:TEvent=TEvent(tData)
```

We then need to check that the event coming in is OUR event, ie the source is from MyCanvas and the event is a gadgetpaint event.

```
If Event.source=MyCanvas And Event.ID=EVENT_GADGETPAINT
```

If that is the case, we can then go ahead and redraw our canvas.

```
                    UpdateCanvas()
```

Remember that what we have done was hijack an event which was enroute to somewhere else, possibly even our WaitEvent() function. In order for this gadgetpaint event not to be executed twice we return a **null** so that the next function in line will not update our canvas anymore.

```
                        Return Null
```

If it was not our event, we must then send back the object tData that came through our function via our hijack as if nothing has happened. The next function in line can then process this event.

```
                     Return tData
```

Phew. That's ends the explanation. I hope that was clear.

Just to reaffirm what we have learnt, lets us add a second hook function called **SecondHook**.

```
SuperStrict

Local x:Int=(GadgetWidth(Desktop())-400)/2
Local y:Int=(GadgetHeight(Desktop())-400)/2

Local MyWindow:TGadget=CreateWindow("EventHook Example", x,y,400,400)
Global MyCanvas:TGadget=CreateCanvas(0,0,380,360,MyWindow)

Local SmallWindow:TGadget=CreateWindow("Move Me", x+125, y+150, 150, 100,
MyWindow, WINDOW_TITLEBAR)

AddHook EmitEventHook, MyHook
AddHook EmitEventHook, SecondHook

Repeat
  WaitEvent()
  Select EventID()
  Case EVENT_WINDOWCLOSE
     End
  Case EVENT_GADGETPAINT
    UpDateCanvas()
   End Select
Forever
End

Function SecondHook:Object(iId:Int,tData:Object,tContext:Object)
  Local Event:TEvent=TEvent(tData)

  If event=Null Return Null

  If Event.source=MyCanvas And Event.ID=EVENT_MOUSEDOWN
     Notify "Click Me"
     Return Null
  EndIf

  Return tData
End Function

Function MyHook:Object(iId:Int,tData:Object,tContext:Object)
  Local Event:TEvent=TEvent(tData)

  If Event.source=MyCanvas And Event.ID=EVENT_GADGETPAINT
```

```
        UpdateCanvas()
        Return Null
      EndIf

      Return tData
    End Function

    Function UpdateCanvas:Int()
        SetGraphics CanvasGraphics(MyCanvas)
        Cls
        SetColor Rnd(255),Rnd(255),Rnd(255)
        DrawRect 10,10,100,150
        Flip
    End Function
```

Apart from the extra **AddHook** statement, we also need to have another hook function. Note the name change and the check for whether the event that came in was a null event or not. By right we also need this check in our earlier MyHook function as well.

```
      Function SecondHook:Object(iId:Int,tData:Object,tContext:Object)
        Local Event:TEvent=TEvent(tData)

        If event=Null Return Null
```

We then check for a mouseclick on our canvas and act accordingly.

```
        If Event.source=MyCanvas And Event.ID=EVENT_MOUSEDOWN
          Notify "Click Me"
          Return Null
        EndIf
```

In the scenario above it would be better to have just a single hook function and do a select/case to check for the various events but this is just to illustrate the possibility of having more than one hook functions in series.

More Readings
There have been some interesting threads on eventhooks but unfortunately most of the examples are complex although one or two were simple enough to learn from.

If you want to learn more, here are some links:-

- *BlitzMax Forums/BlitzMax Beginners Area/MaxGUI menu over a canvas* - This thread was the one that gave me the idea on how to approach this tutorial (link)
- *BlitzMax Forums/BlitzMax Beginners Area/Events / Hooks and Stuff* - This has a explanation by Mark Sibly himself on Eventhooks (see below), worth reading. Also check out the slider code by René as well as the by Pertubatio at the end (link)
- *BlitzMax Forums/BlitzMax Beginners Area/events* - This got me confused but may be worth a read (link)
- *BlitzMax Forums/BlitzMax Programming/MaxGui Program Flow* - A proposal on how to use eventhooks with gadgets and program flow. Advance stuff (link)

This was what Mark wrote about eventhooks
A couple of points:

** The 'MyEventHook' function is the 'hook' and gets called *every time* an event occurs (from within WaitEvent - or more precisely, WaitSystem, which is called by WaitEvent).*

** Hooks should check the 'data' param to make sure it's valid. In this example it always will be, but it's a good idea in general.*

** Hooks should generally return the 'data' param they are passed - this is so other hooks can look at the data too. So if you want to 'block' other hooks, return Null.*

Both polled input (KeyDown, KeyHit etc) and the event queue are built on top of hooks - check out the source code to really confuse yourselves!

Why use hooks? Why not just stick with WaitEvent? Well, hooks are mainly useful for dealing with 'modal actions' - actions which involve the user holding down the mouse and doing something, like resizing a window or dragging a scroller.

When the user is performing a modal action, the OS goes into a loop and notifies your app of events via a 'callback function'. Hooks are really just a generalization of such callback functions.

BlitzPlus got around this using a complex system of mirrors and pulleys, but I was nervous about using the same trick on multiple platforms, much to skid's disappointment.

Hope this helps a bit!

Summary

I am sure that there are many clever things that can be done with eventhooks. But at the simplest level, it is nothing more than a means to hijack (hook into) the event system to allow the programmer to deal with 'modal' situation.

To setup an eventhook, we need to

- Issue an **addhook** statement, telling MaxGUI which function to hook to
- write the hook function, being careful to follow certain protocols
- Note the MaxGUI rules on events, the events needs to be generated for eventhooks to catch them (for example, when using panels, set the PANEL_ACTIVE style so that events are generated)

That's all there is to it in the simplest sense. One can then encapsulate this concept into types and OOPs and other stuff as well of course :)

That's all for now. Return to Main Index.