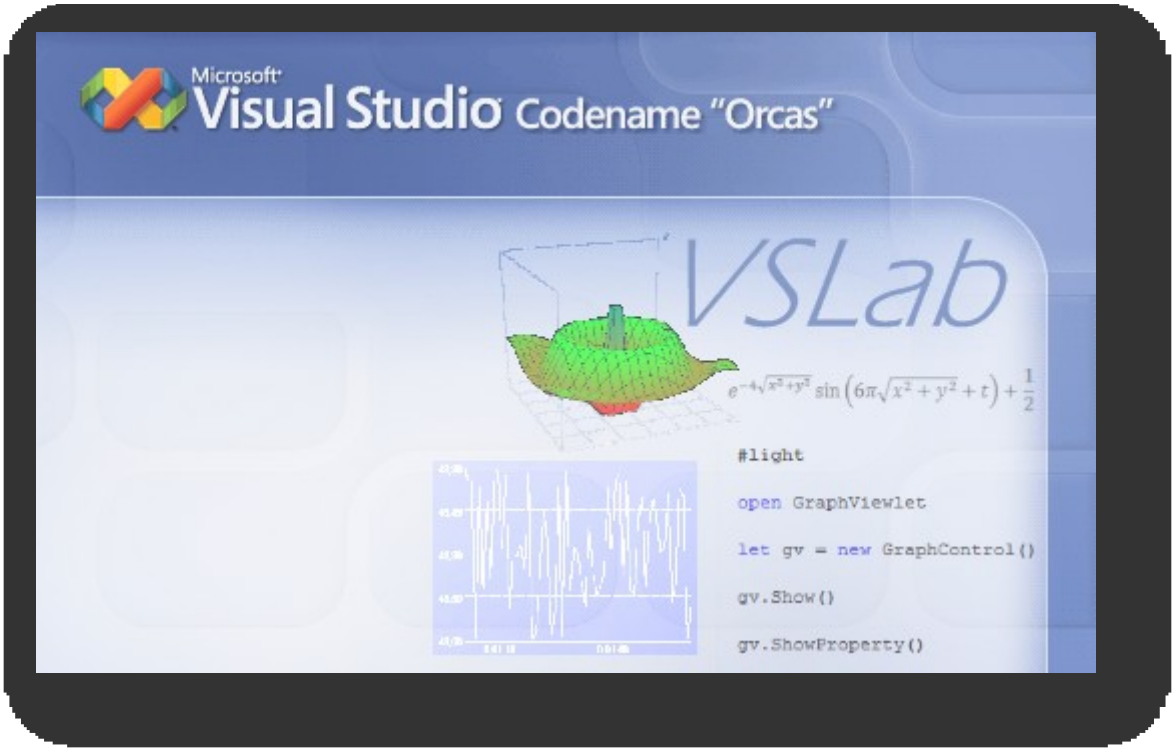
How to write a Viewlet

Whitepaper



In this paper we introduce the core notion of Viewlet and we discuss how to implement Viewlets for VSLab.

Author: Antonio Cisternino (cisterni@di.unipi.it), University of Pisa
Version: 1.0
Last update: 6/28/2008 8:40:00 PM

Introduction

In this *let epoch we couldn't resist the temptation of calling the core elements of VSLab *Viewlet*. A *Viewlet* is simply a Visual Studio toolwindow whose component is defined inside F# interactive. Although it may seem a little difference from standard toolwindows it is a big one: the state of the *Viewlet* resides in the *fsi.exe* process rather than in *devenv.exe* allowing quick access to the whole environment of F# interactive. Thus we can easily exchange data (including functions, since F# is a functional programming language) between scripts and viewlets, with the addition of a natural integration with the Visual Studio windowing system.

Viewlets can be written in any .NET language as long as the assembly is made available to F# interactive using the #r directive. You can also write viewlets on the fly by evaluate their definition and instantiate them interactively as shown in the Visual tutorial.

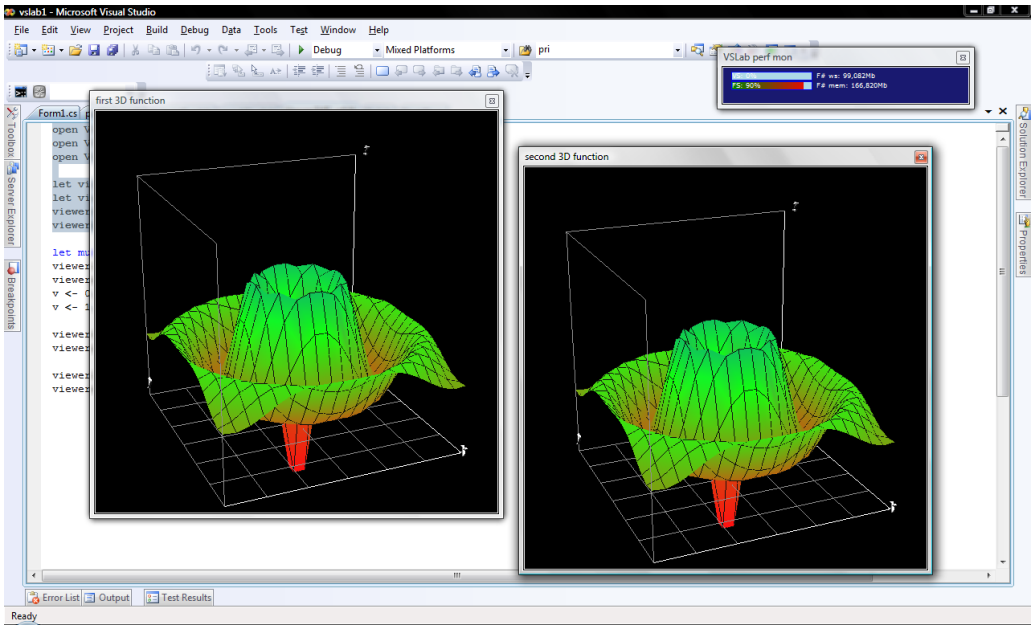
Note. In this How To we assume you are familiar with F# and Windows Forms, there are several resources available to learn F#, including "*Expert F#*" a book I co-authored from APress. F# is a wonderful language and Somasegar has announced in November 2007 that it will be a first programming language from Microsoft, thus it is a good investment if you learn it.

Anatomy of a Viewlet

We have worked really hard to make Viewlets as transparent as possible so that you can leverage on your existing Windows Forms code and knowledge. A Viewlet is defined by a class that inherits from *Viewlet* and it is a Windows Form user control with the additional constraint that you cannot have children controls. Writing Viewlets is a good exercise of graphical controls writing.

Note. If you are interested in the implementation details have a look to the whitepaper dedicated to VSLab implementation.

In this tutorial we discuss the implementation of VSMon a viewlet designed for monitoring Visual Studio and F# CPU usage and F# interactive memory load.



Structure of the Viewlet

The basic structure of the viewlet is the following:

```
#light

open VSLabFSICore
open VSLabViewlets

type VSMon() as x =
  inherit Viewlet() as base
  override x.OnPaint(e) = (* … *)
```

As you can see we are defining a class that inherits from *Viewlet* and usually overrides the *OnPaint* method which performs the drawing required by our Viewlet. Since we are interested in showing CPU and memory consumption we used the *System.Diagnostics.Process* class abilities to retrieve this information:

```
  let mutable lastRead = DateTime.Now
  let mutable vscput = new TimeSpan()
  let mutable fsicput = new TimeSpan()
  let mutable vsCPU = 0
  let mutable fsCPU = 0
  let mutable wsSz = 0f
  let mutable msSz = 0f

  let t = new Timer()

  let CPUTime () =
    let vp = Process.GetProcessById(Viewlets.VSPID)
    let fp = Process.GetCurrentProcess()
    let now = DateTime.Now
    vsCPU <- Math.Min(int (100.0 * (vp.TotalProcessorTime -
vscput).TotalMilliseconds / (now - lastRead).TotalMilliseconds), 100)
    fsCPU <- Math.Min(int (100.0 * (fp.TotalProcessorTime -
fsicput).TotalMilliseconds / (now - lastRead).TotalMilliseconds), 100)
    lastRead <- now
    vscput <- vp.TotalProcessorTime
    fsicput <- fp.TotalProcessorTime
    msSz <- float32(fp.PrivateMemorySize64) / (1024f*1024f)
    wsSz <- float32(fp.WorkingSet64)  / (1024f*1024f)
```

A call to *CPUTime* updates the values and we can update the displayed data. The data is updated by a timer that is declared in the constructor, and we also set some property of the control:

```
  let mutable showMemory = true
  let Title = "VSLab perf mon"
  let t = new Timer()

  do
    x.BackColor <- Color.Navy
    x.Font <- new Font("Verdana", 6f)
    x.Name <- Title

    for v in Viewlets.Items do
      if v.Name = Title then
        failwith "Only one instance of Perf mon is allowed!"

    t.Interval <- 500
    t.Tick.Add(fun _ -> CPUTime(); x.Invalidate())
```

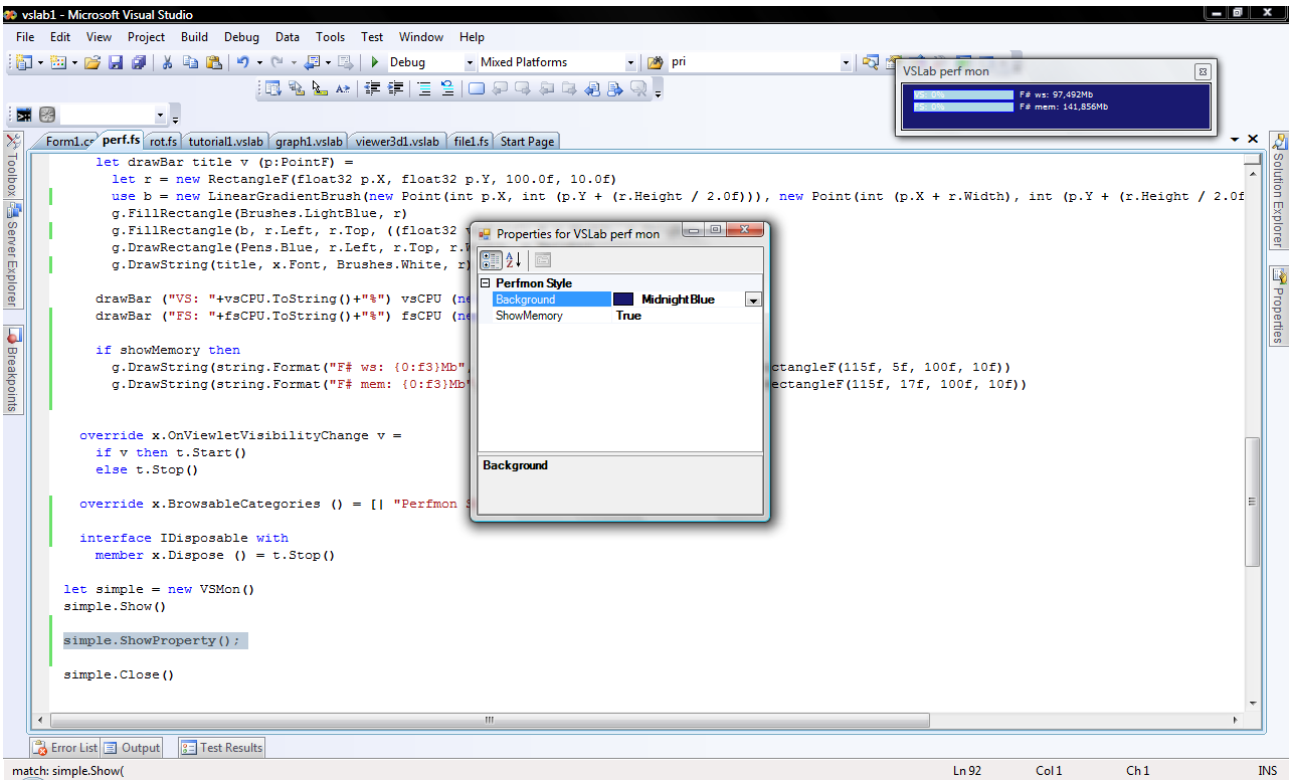
The *showMemory* flag will be used to decide whether memory information should be displayed or not. The timer is set to an interval of 500 milliseconds and it simply update counters and invalidates the control. Note that we do not start the timer since it will be started only when the viewlet becomes visible.

Note. Visual Studio and F# interactive are powerful tools, but it is important to use resource wisely, if you use timers for visualzation remember to stop and start them upon visibility change of the viewlet.

The *OnViewletVisibilityChanged* event informs the viewlet about its visibility state, we rely on this notification in order to start and stop the update timer:

```
override x.OnViewletVisibilityChange v =
  if v then t.Start()
  else t.Stop()
```

Viewlets inherit the *ShowProperty* method which displays a dialog with a property grid to configure the running viewlet.



You can control properties displayed by the grid by overriding the *BrowsableCategories* property:

```
override x.BrowsableCategories () = [| "Perfmon Style" |]
```

Then you simply annotate using custom attributes the properties you want to expose, in our example the background and whether the memory usage should be displayed or not:

```
[<Category("Perfmon Style")>]
member x.Background
  with get() = x.BackColor
  and set c = x.BackColor <- c

[<Category("Perfmon Style")>]
member x.ShowMemory
  with get() = showMemory
```

```
    and set m = showMemory <- m; x.Invalidate()
```

The *OnPaint* method simply draws two bars for CPU load and display strings for memory consumption. It is a standard *paint* method of a WinForms user control:

```
  override x.OnPaint(e) =
    let g = e.Graphics
    let drawBar title v (p:PointF) =
      let r = new RectangleF(float32 p.X, float32 p.Y, 100.0f, 10.0f)
      use b = new LinearGradientBrush(new Point(int p.X, int (p.Y + (r.Height /
2.0f))), new Point(int (p.X + r.Width), int (p.Y + (r.Height / 2.0f))),
Color.Green, Color.Red )
      g.FillRectangle(Brushes.LightBlue, r)
      g.FillRectangle(b, r.Left, r.Top, ((float32 v)/100f) * r.Width, r.Height)
      g.DrawRectangle(Pens.Blue, r.Left, r.Top, r.Width, r.Height)
      g.DrawString(title, x.Font, Brushes.White, r)

    drawBar ("VS: "+vsCPU.ToString()+"%") vsCPU (new PointF(10.0f, 5.0f))
    drawBar ("FS: "+fsCPU.ToString()+"%") fsCPU (new PointF(10.0f, 17.0f))

    if showMemory then
      g.DrawString(string.Format("F# ws: {0:f3}Mb", wsSz), x.Font,
Brushes.White, new RectangleF(115f, 5f, 100f, 10f))
      g.DrawString(string.Format("F# mem: {0:f3}Mb", msSz), x.Font,
Brushes.White, new RectangleF(115f, 17f, 100f, 10f))
```

Resources allocated by a Viewlet must be disposed when it gets closed using the *Close* method (or upon Visual Studio termination). You can simply implement the *IDisposable* interface:

```
  interface IDisposable with
    member x.Dispose () = t.Stop()
```

Conclusions

Writing Viewlets it is essentially equivalent to developing WinForms controls, though the actual implementation is quite complex. In particular you must not rely on timings typical of Windows Forms since the viewlet performs drawings and receives events from a peer control running inside Visual Studio. You cannot also rely on all the events of a standard control, check the *Messages* enumeration to see what events are routed to the Viewlet. A lightweight control library is being developed in order to support child controls in Viewlets.

Appendix: the full source code of VSMon Viewlet

```
#light

open System
open System.ComponentModel
open System.Drawing
open System.Drawing.Drawing2D
open System.Windows.Forms
open System.Diagnostics

open VSLabFSICore
open VSLabViewlets

type VSMon() as x =
  inherit Viewlet() as base
```

```
  let mutable lastRead = DateTime.Now
  let mutable vscput = new TimeSpan()
  let mutable fsicput = new TimeSpan()
  let mutable vsCPU = 0
  let mutable fsCPU = 0
  let mutable wsSz = 0f
  let mutable msSz = 0f
  let mutable showMemory = true

  let Title = "VSLab perf mon"

  let t = new Timer()

  let CPUTime () =
    let vp = Process.GetProcessById(Viewlets.VSPID)
    let fp = Process.GetCurrentProcess()
    let now = DateTime.Now
    vsCPU <- Math.Min(int (100.0 * (vp.TotalProcessorTime -
vscput).TotalMilliseconds / (now - lastRead).TotalMilliseconds), 100)
    fsCPU <- Math.Min(int (100.0 * (fp.TotalProcessorTime -
fsicput).TotalMilliseconds / (now - lastRead).TotalMilliseconds), 100)
    lastRead <- now
    vscput <- vp.TotalProcessorTime
    fsicput <- fp.TotalProcessorTime
    msSz <- float32(fp.PrivateMemorySize64) / (1024f*1024f)
    wsSz <- float32(fp.WorkingSet64)  / (1024f*1024f)

  do
    x.BackColor <- Color.Navy
    x.Font <- new Font("Verdana", 6f)
    x.Name <- Title

    for v in Viewlets.Items do
      if v.Name = Title then failwith "Only one instance of Perf mon is
allowed!"

    t.Interval <- 500
    t.Tick.Add(fun _ -> CPUTime(); x.Invalidate())

  [<Category("Perfmon Style")>]
  member x.Background
    with get() = x.BackColor
    and set c = x.BackColor <- c

  [<Category("Perfmon Style")>]
  member x.ShowMemory
    with get() = showMemory
    and set m = showMemory <- m; x.Invalidate()

  override x.OnPaint(e) =
    let g = e.Graphics
    let drawBar title v (p:PointF) =
      let r = new RectangleF(float32 p.X, float32 p.Y, 100.0f, 10.0f)
      use b = new LinearGradientBrush(new Point(int p.X, int (p.Y + (r.Height /
2.0f))), new Point(int (p.X + r.Width), int (p.Y + (r.Height / 2.0f))),
Color.Green, Color.Red )
      g.FillRectangle(Brushes.LightBlue, r)
      g.FillRectangle(b, r.Left, r.Top, ((float32 v)/100f) * r.Width, r.Height)
      g.DrawRectangle(Pens.Blue, r.Left, r.Top, r.Width, r.Height)
      g.DrawString(title, x.Font, Brushes.White, r)

    drawBar ("VS: "+vsCPU.ToString()+"%") vsCPU (new PointF(10.0f, 5.0f))
    drawBar ("FS: "+fsCPU.ToString()+"%") fsCPU (new PointF(10.0f, 17.0f))
```

```
    if showMemory then
      g.DrawString(string.Format("F# ws: {0:f3}Mb", wsSz), x.Font,
Brushes.White, new RectangleF(115f, 5f, 100f, 10f))
      g.DrawString(string.Format("F# mem: {0:f3}Mb", msSz), x.Font,
Brushes.White, new RectangleF(115f, 17f, 100f, 10f))


  override x.OnViewletVisibilityChange v =
    if v then t.Start()
    else t.Stop()

  override x.BrowsableCategories () = [| "Perfmon Style" |]

  interface IDisposable with
    member x.Dispose () = t.Stop()
```