

Forum:[Hack Forums](#)

Thread: [Custom GDI Widgets Base on Desktop concept](#)

TheBarret - September 3, 2014, 12:07 pm

Been while ive made something new, maybe few of you ever used Rainmeter,
Its basicly what the built-in windows widget manager does but then with a open front-end
for developers to create their own style widget that returns or shows certain info.

i always wanted to make my own kind of widget but not something that would eat up my
CPU/memory.

Now i know GDI is very hungry for memory but we can suppress its hunger by
implementing a simple on change only update, so we only draw if something is doing something
and it should
suppress alot of unneeded draw calls, lets continue...

This code is NOT yet finished but the widget controller already works as intended so altho its not
finished
this base can be adapted very easily and used in various situations to ur needs.

Alright continue....

I took a basic form i disabled the border/style so we end up with a naked form with no border
then i change the window state to maximized and lets set the ShowIntaskbar to false.
Don't worry the entire form will not intercept the clicks for your desktop icons.

We apply a few styles to overcome some flickering issues and we enable the
'TransparentBackColor' option
to accomplish the transparent drawing.

Lets list all the things i want to do:

- I want be able to drag widgets (forms)
- I want a nice controller for all wigdets (Container)
- I want to distinct widget render priority (ZOrder)
- And i want some kind of auto-alignment (screen border pull and let go)

For the form_load setup:

Code Public Class FrmMain

Private GUI As Manager

Private Sub FrmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load

Me.SetStyle(ControlStyles.UserPaint, True)

Me.SetStyle(ControlStyles.AllPaintingInWmPaint, True)

Me.SetStyle(ControlStyles.OptimizedDoubleBuffer, True)

Me.SetStyle(ControlStyles.SupportsTransparentBackColor, True)

Me.GUI = New Manager(Me)

End Sub

End Class

Draw Manager:

Code Option Strict On

Public Class Manager

Implements IDisposable

Private Property Parent As Form

Private Property ActiveWindow As Window

Private Property DragOffsetX As Integer

Private Property DragOffsetY As Integer

Private Property Order As ZOrder

Private Property PullOffset As Margin

Private Property MonitorOffsets As Margin

Private Property IsDragging As Boolean

Private Property IsUpdate As Boolean

Private Property Container As Container

Sub New(Parent As Form)

Me.Parent = Parent

Me.Parent.TransparencyKey = Me.Parent.BackColor

Me.Parent.WindowState = FormWindowState.Maximized

Me.Parent.ShowInTaskbar = False

Me.IsDragging = False

Me.IsUpdate = True

Me.MonitorOffsets = New Margin() With {.Left = 8, .Right = 6, .Top = 8, .Bottom = 0}

Me.PullOffset = New Margin() With {.Left = 200, .Right = 300, .Top = 200, .Bottom = 300}

Me.Parent.KeyPreview = True

Me.Order = New ZOrder

Me.Container = New Container

AddHandler Me.Parent.Paint, AddressOf Me.EventPaint

AddHandler Me.Parent.MouseMove, AddressOf Me.EventMouseMove

AddHandler Me.Parent.MouseDown, AddressOf Me.EventMouseDown

AddHandler Me.Parent.MouseUp, AddressOf Me.EventMouseUp

End Sub

Public Sub CreateWindow(Name As String, Title As String, X As Integer, Y As Integer, Width As Integer, Height As Integer)

Dim T As Window = New Window(Name, X, Y, Width, Height) With {.Title = Title}

Me.Container.Add(T)

Me.Order.Insert(0, T)

Me.Update()

End Sub

Public Sub Update()

Me.IsUpdate = True

Me.Parent.Invalidate()

End Sub

Private Sub EventMouseMove(sender As Object, e As MouseEventArgs)

If (Me.IsDragging) And (Not Me.ActiveWindow Is Nothing) Then

If (e.X <= Me.PullOffset.Left) AndAlso

(e.Y <= Me.PullOffset.Top) Then

Me.ActiveWindow.X = Me.MonitorOffsets.Left

Me.ActiveWindow.Y = Me.MonitorOffsets.Top

Elseif (e.X >= (SystemInformation.PrimaryMonitorSize.Width - Me.PullOffset.Right))

AndAlso

(e.Y >= (SystemInformation.PrimaryMonitorSize.Height - Me.PullOffset.Bottom))

Then

Me.ActiveWindow.X = SystemInformation.PrimaryMonitorSize.Width -

```

Me.ActiveWindow.Width + Me.MonitorOffsets.Right
    Me.ActiveWindow.Y = SystemInformation.PrimaryMonitorSize.Height -
Me.ActiveWindow.Height + Me.MonitorOffsets.Bottom
    Elself (e.X <= Me.PullOffset.Left) AndAlso
        (e.Y >= (SystemInformation.PrimaryMonitorSize.Height - Me.PullOffset.Bottom))
Then
    Me.ActiveWindow.X = Me.MonitorOffsets.Left
    Me.ActiveWindow.Y = SystemInformation.PrimaryMonitorSize.Height -
Me.ActiveWindow.Height + Me.MonitorOffsets.Bottom
    Elself (e.X >= (SystemInformation.PrimaryMonitorSize.Width - Me.PullOffset.Right))
AndAlso
    (e.Y <= Me.PullOffset.Top) Then
        Me.ActiveWindow.X = SystemInformation.PrimaryMonitorSize.Width -
Me.ActiveWindow.Width + Me.MonitorOffsets.Right
        Me.ActiveWindow.Y = Me.MonitorOffsets.Top
    Elself (e.X <= Me.PullOffset.Left) Then
        Me.ActiveWindow.X = Me.MonitorOffsets.Left
        Me.ActiveWindow.Y = e.Y - Me.DragOffsetY
    Elself (e.X >= (SystemInformation.PrimaryMonitorSize.Width - Me.PullOffset.Right)) Then
        Me.ActiveWindow.X = SystemInformation.PrimaryMonitorSize.Width -
Me.ActiveWindow.Width + Me.MonitorOffsets.Right
        Me.ActiveWindow.Y = e.Y - Me.DragOffsetY
    Elself (e.Y <= Me.PullOffset.Top) Then
        Me.ActiveWindow.X = e.X - Me.DragOffsetX
        Me.ActiveWindow.Y = Me.MonitorOffsets.Top
    Elself (e.Y >= (SystemInformation.PrimaryMonitorSize.Height - Me.PullOffset.Bottom))
Then
    Me.ActiveWindow.X = e.X - Me.DragOffsetX
    Me.ActiveWindow.Y = SystemInformation.PrimaryMonitorSize.Height -
Me.ActiveWindow.Height + Me.MonitorOffsets.Bottom
Else
    Me.ActiveWindow.X = e.X - Me.DragOffsetX
    Me.ActiveWindow.Y = e.Y - Me.DragOffsetY
End If
Me.Update()
End If
End Sub

Private Sub EventMouseDown(sender As Object, e As MouseEventArgs)
    Me.ActiveWindow = Me.Order.GetWindowAt(e.X, e.Y)
    If (Not ActiveWindow Is Nothing) Then
        If (Me.Order.Count > 0) AndAlso
            (Me.Order.FirstOrDefault IsNot Me.ActiveWindow) Then
            Me.Order.Swap(Me.ActiveWindow)
            Me.Update()
        End If
        Me.DragOffsetX = e.X - Me.ActiveWindow.X
        Me.DragOffsetY = e.Y - Me.ActiveWindow.Y
        Me.IsDragging = True
        Me.ActiveWindow.IsDragging = True
        Me.Update()
    End If
End Sub

Private Sub EventMouseUp(sender As Object, e As MouseEventArgs)

```

```

    If (Me.IsDragging) Then
        Me.IsDragging = False
        Me.ActiveWindow = Nothing
        Me.Container.Reset()
    End If
    Me.Update()
End Sub
Private Sub EventPaint(sender As Object, e As PaintEventArgs)
    If (Me.IsUpdate) Then
        Using Bm As New Bitmap(Me.Parent.Width, Me.Parent.Height)
            Using g As Graphics = Graphics.FromImage(Bm)
                g.Clear(Color.Transparent)
                g.InterpolationMode = Drawing2D.InterpolationMode.HighQualityBilinear
                g.SmoothingMode = Drawing2D.SmoothingMode.AntiAlias
                g.Flush()
                Me.Order.Render(g)
            End Using
            Me.Parent.BackgroundImage = DirectCast(Bm.Clone, Image)
        End Using
        Me.IsUpdate = False
    End If
End Sub
#Region "IDisposable Support"
Private disposedValue As Boolean
Protected Overridable Sub Dispose(disposing As Boolean)
    If Not Me.disposedValue Then
        If disposing Then
            Me.Container.Clear()
            Me.Container = Nothing
            RemoveHandler Me.Parent.Paint, AddressOf Me.EventPaint
            RemoveHandler Me.Parent.MouseMove, AddressOf Me.EventMouseMove
            RemoveHandler Me.Parent.MouseDown, AddressOf Me.EventMouseDown
            RemoveHandler Me.Parent.MouseUp, AddressOf Me.EventMouseUp
        End If
    End If
    Me.disposedValue = True
End Sub
Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub
#End Region
End Class
Remark:
- MonitorOffset might be variable PER gfx card or type of monitor, adjust them to your settings

ZOrder Class:
Code Public Class ZOrder
    Inherits List(Of Window)
    Public Sub Swap(W As Window)
        Me(Me.IndexOf(W)) = Me(0) : Me.Insert(0, W)
    End Sub
    Public Sub Render(G As Graphics)
        For i As Integer = Me.Count - 1 To 0 Step -1

```

```

        Me(i).Draw(G)
    Next
End Sub
Public Function GetWindowAt(X As Integer, Y As Integer) As Window
    For i As Integer = 0 To Me.Count - 1
        If (Me(i).CollisionTest(X, Y)) Then Return Me(i)
    Next
    Return Nothing
End Function
End Class

```

The Window Container:

```

Code Public Class Container
    Inherits CollectionBase
    Public Function Add(ByVal Wnd As Window) As Integer
        Return Me.List.Add(Wnd)
    End Function
    Default Public ReadOnly Property Index(ByVal i As Integer) As Window
        Get
            Return Me.List(i)
        End Get
    End Property
    Public Sub Reset()
        For Each W As Window In Me
            W.IsDragging = False
        Next
    End Sub
    Public Sub Draw(ByVal G As Graphics)
        For Each Wnd As Window In Me
            Wnd.Draw(G)
        Next
    End Sub
End Class

```

Now for the widget container itself (acts like a window):

```

Code Option Strict On
Public Class Window
    Public Property Name As String
    Public Property X As Integer
    Public Property Y As Integer
    Public Property Font As Font
    Public Property Width As Integer
    Public Property Height As Integer
    Public Property IsDragging As Boolean
    Public Property BodyColor As Color
    Public Property BorderColor As Color
    Public Property HeaderColor As Color
    Public Property CaptionColor As Color
    Public Property CaptionHeight As Integer
    Public Property Title As String
    Public Property Transparency As Integer
    Sub New(Name As String, X As Integer, Y As Integer, W As Integer, H As Integer)
        Me.Name = Name
        Me.X = X
        Me.Y = Y
    End Sub

```

```

Me.Width = W
Me.Height = H
Me.Title = String.Empty
Me.CaptionHeight = 25
Me.IsDragging = False
Me.Transparency = 100
Me.HeaderColor = Color.CornflowerBlue
Me.BodyColor = Color.White
Me.BorderColor = Color.FromArgb(Me.Transparency, 0, 0, 0)
Me.CaptionColor = Color.Black
Me.Font = New Font("Lucida Console", 8)
End Sub
Public Sub Draw(G As Graphics)
    G.FillRectangle(DirectCast(If(Me.IsDragging, New
SolidBrush(Color.FromArgb(Me.Transparency, Color.CornflowerBlue.R, Color.CornflowerBlue.G,
Color.CornflowerBlue.B)), Brushes.CornflowerBlue), Brush), Me.X, Me.Y, Me.Width,
Me.CaptionHeight)
    G.DrawRectangle(New Pen(Me.BorderColor, 1), Me.X, Me.Y, Me.Width, Me.CaptionHeight)
    G.FillRectangle(DirectCast(If(Me.IsDragging, New
SolidBrush(Color.FromArgb(Me.Transparency, Me.BodyColor.R, Me.BodyColor.G,
Me.BodyColor.B)), Brushes.White), Brush), Me.X, Me.Y + Me.CaptionHeight, Me.Width, Me.Height -
Me.CaptionHeight)
    G.DrawRectangle(New Pen(Me.BorderColor, 1), Me.X, Me.Y + Me.CaptionHeight,
Me.Width, Me.Height - Me.CaptionHeight)
    G.DrawString(String.Format("{0}", Me.Title), Me.Font, New SolidBrush(Me.CaptionColor),
Me.X + 8, Me.Y + 8)
End Sub
Public Function CollisionTest(ByVal X As Integer, ByVal Y As Integer) As Boolean
    Return ((X >= Me.X) AndAlso (X <= Me.X + Me.Width)) AndAlso
        ((Y >= Me.Y) AndAlso (Y <= Me.Y + Me.Height))
End Function
Public Function GetDistance(ByVal X As Integer, ByVal Y As Integer) As Double
    Dim DistX As Double = Math.Abs(Me.Center.X - X)
    Dim DistY As Double = Math.Abs(Me.Center.Y - Y)
    If ((DistX + DistY) = 0) Then Return 0 Else Return (DistX + DistY) / 2
End Function
Public ReadOnly Property Center As Point
    Get
        Return New Point(Me.X + Me.Width \ 2, Me.Y + Me.Height \ 2)
    End Get
End Property
End Class

```

Ok we can understand all that, dont we? Lets continue,

Now lets test one of these options ive named so far:

The Render prioity in three stages (a click and hold click = transparent effect):
(Click&Drag makes widgets transparent for a nice effect)

Now for the auto-alignment, i can't show you this by a GIF recorder since the GDI drawn on desktop does not capture this particulaire part, anyway what i can do is explain the code.

I made a margin structure that has 4 integer values, left, right, top and bottom.

(Remark: this code only supports one monitor setup, if you want more change the code urself)

Code Public Structure Margin

Public Right As Integer

Public Left As Integer

Public Top As Integer

Public Bottom As Integer

End Structure

These values define the auto-alignment with the monitor border, they are located in the Manager class / EventMouseMove .

Basicly when we are dragging an object we check the X or Y pointer and if it becomes lower or higher then our margins

then we set X or Y of that object to a fixed value, therefore it will auto-pull the window in all directions, also corners included.

we end up with this crude base:

Spoiler [\(Click to View\)](#)

Testing a few demo wigdets :

Code Me.GUI.CreateWindow("#1", "Widget Container #1", 100, 100, 200, 200)

Me.GUI.CreateWindow("#2", "Widget Container #3", 200, 200, 200, 200)

Me.GUI.CreateWindow("#3", "Widget Container #2", 300, 300, 200, 200)For now its all a raw basic concept but i want to extend this and on the end i want scriptable "widgets" that have their own Call events and such, custom clocks, Facebook API or GMail API or system information in visual render.

Suggestions or idea's welcome...

- Barret

\$lick - September 3, 2014, 12:13 pm

Another great release from DJ Black Barret.

You also have a good taste in games.

SaketSaket - September 3, 2014, 12:37 pm

Nice release & something unique... ;)

Cornflower Blue ???-?— - September 3, 2014, 1:12 pm

Why don't you use DirectX or OpenGL? GDI is bad unless you can optimize it well enough

SIM-1847363 - September 3, 2014, 1:47 pm

Seems something i should try out, thanks for the tutorial, nice release

SIM-1456245 - September 3, 2014, 3:12 pm

I've seen this, but I now understand that it has NOTHING to do with any of your scripting langs. Great job tho.

nuk - September 3, 2014, 5:48 pm

THANK YOU! Was looking for something like this a few weeks back ;)