

Forum:[Hack Forums](#)

Thread: [Basic Windows Shell](#)

Blixuk - October 29, 2014, 4:57 am

So I hacked this together using a few different things. It works, but not beautifully at all. I thought I would share it and see if anyone could clean it up and improve on the whole thing.

This is a basic Windows Shell.

It can display all open windows and allow you select and open them as well as allowing you to minimizes them and bring them back.

This also removes the current window shell and makes a new shell location at the top of the screen. This also allows for windows to full screen without over lapping your shell.

You will need two classes that I will add here one for the Windows API and another for the Functions.

Functions.

Code Imports System.Windows.Forms

Imports System.Diagnostics

Imports System.Collections

Public Class Functions

Structure WindowData

Public hwnd As IntPtr

Public title As String

End Structure

#Region "Private variables"

Private Shared m_rcOldDesktopRect As WinAPI.RECT

Private Shared m_hTaskBar As IntPtr

#End Region

'Resizes the Desktop area to our shells' requirements

Public Shared Sub MakeNewDesktopArea()

' Save current Working Area size

m_rcOldDesktopRect.left = SystemInformation.WorkingArea.Left

m_rcOldDesktopRect.top = SystemInformation.WorkingArea.Top

m_rcOldDesktopRect.right = SystemInformation.WorkingArea.Right

m_rcOldDesktopRect.bottom = SystemInformation.WorkingArea.Bottom

'Make a new Workspace

Dim rc As WinAPI.RECT

rc.left = SystemInformation.VirtualScreen.Left

'We reserve the 24 pixels on top for our taskbar

rc.top = SystemInformation.VirtualScreen.Top + 25

rc.right = SystemInformation.VirtualScreen.Right

rc.bottom = SystemInformation.VirtualScreen.Bottom

WinAPI.SystemParametersInfo(CInt(WinAPI.SPI.SPI_SETWORKAREA), 0, rc, 0)

End Sub

'Restores the Desktop area

Public Shared Sub RestoreDesktopArea()

WinAPI.SystemParametersInfo(CInt(WinAPI.SPI.SPI_SETWORKAREA), 0,
m_rcOldDesktopRect, 0)

End Sub

'Hides the Windows Taskbar

Public Shared Sub HideTaskBar()

' Get the Handle to the Windows Taskbar

m_hTaskBar = WinAPI.FindWindow("Shell_TrayWnd", Nothing)

' Hide the Taskbar

If CInt(m_hTaskBar) <> 0 Then

WinAPI.ShowWindow(m_hTaskBar, CInt(WinAPI.WindowShowStyle.Hide))

End If

End Sub

'Show the Windows Taskbar

Public Shared Sub ShowTaskBar()

If CInt(m_hTaskBar) <> 0 Then

WinAPI.ShowWindow(m_hTaskBar, CInt(WinAPI.WindowShowStyle.Show))

End If

End Sub

'Gets a list of Active Tasks

Public Shared Function GetActiveTasks() As ArrayList

Dim ar As New ArrayList()

Dim child As IntPtr = IntPtr.Zero

Dim process__1 As Process() = Process.GetProcesses()

For Each p As Process In process__1

Dim w As WindowData

If p.MainWindowHandle <> IntPtr.Zero AndAlso p.MainWindowTitle.Length > 0 Then

w.hwnd = p.MainWindowHandle

w.title = p.MainWindowTitle

ar.Add(w)

End If

Next

Return ar

ar.Clear()

End Function

End Class

WinAPI.

Code Imports System.Text

Imports System.Runtime.InteropServices

Public Class WinAPI

Public Structure RECT

Public left As Integer

Public top As Integer

```
Public right As Integer
Public bottom As Integer
End Structure
```

```
Public Structure PointAPI
Public x As Integer
Public y As Integer
End Structure
```

```
Public Structure WindowPlacement
Public Length As Integer
Public Flags As Integer
Public showCmd As Integer
Public MinPosition As PointAPI
Public MaxPosition As PointAPI
Public NormalPosition As RECT
End Structure
```

```
'ShowWindow
Public Enum WindowShowStyle As Integer
Hide = 0
ShowNormal = 1
ShowMinimized = 2
ShowMaximized = 3
Maximize = 3
ShowNormalNoActivate = 4
Show = 5
Minimize = 6
ShowMinNoActivate = 7
ShowNoActivate = 8
Restore = 9
ShowDefault = 10
ForceMinimized = 11
End Enum
```

```
Public Const HSHELL_WINDOWCREATED As Integer = 1
Public Const HSHELL_WINDOWDESTROYED As Integer = 2
Public Const HSHELL_WINDOWACTIVATED As Integer = 4
```

```
'SystemParametersInfo
Public Enum SPI As Integer
SPI_SETWORKAREA = &H2F
SPI_GETWORKAREA = &H30
End Enum
```

```
Private Declare Function GetWindowPlacement Lib "user32" (ByVal hwnd As Integer, ByRef lpwndpl As WindowPlacement) As Integer
```

```
<DllImport("user32.dll", SetLastError:=True)> _
Public Shared Function SystemParametersInfo(ByVal uiAction As UInteger, ByVal uiParam As UInteger, ByRef pvParam As RECT, ByVal fWinIni As UInteger) As Boolean
End Function
```

```

<DllImport("user32.dll", SetLastError:=True)> _
Public Shared Function FindWindow(ByVal lpClassName As String, ByVal lpWindowName As
String) As IntPtr
End Function

<DllImport("user32.dll")> _
Public Shared Function ShowWindow(ByVal hWnd As IntPtr, ByVal nCmdShow As Integer) As
Boolean
End Function

<DllImport("user32.dll")> _
Public Shared Function SetWindowPos(ByVal hWnd As IntPtr, ByVal hWndInsertAfter As IntPtr,
ByVal X As Integer, ByVal Y As Integer, ByVal cx As Integer, ByVal cy As Integer, _
ByVal uFlags As UInteger) As Boolean
End Function

<DllImport("user32.dll")> _
Public Shared Function SetForegroundWindow(ByVal hWnd As IntPtr) As
<MarshalAs(UnmanagedType.Bool)> Boolean
End Function

<DllImport("user32.dll", SetLastError:=True, CharSet:=CharSet.Auto)> _
Public Shared Function GetWindowTextLength(ByVal hWnd As IntPtr) As Integer
End Function

<DllImport("user32.dll", CharSet:=CharSet.Auto, SetLastError:=True)> _
Public Shared Function GetWindowText(ByVal hWnd As IntPtr, ByVal lpString As StringBuilder,
ByVal nMaxCount As Integer) As Integer
End Function

<DllImport("user32.dll", SetLastError:=True)> _
Public Shared Sub SwitchToThisWindow(ByVal hWnd As IntPtr, ByVal fAltTab As Boolean)
End Sub

<DllImport("user32.dll")> _
Public Shared Function SetTaskmanWindow(ByVal hWnd As IntPtr) As Boolean
End Function

<DllImport("user32.dll")> _
Public Shared Function RegisterShellHookWindow(ByVal hWnd As IntPtr) As Integer
End Function

<DllImport("user32.dll")> _
Public Shared Function RegisterWindowMessage(ByVal lpString As String) As Integer
End Function

<DllImport("user32.dll")> _
Public Shared Function DeregisterShellHookWindow(ByVal hWnd As IntPtr) As Boolean
End Function

```

End Class

I'm not going to go in detail about them.

Ok, so here's how to use them to create our Shell.

You need 2 Forms One is just a Launcher and isn't required but helps for testing.
The second is the shell itself.

The Launcher is really simple. You don't need anything on this at all so make it really small. Just make sure you can use the exit button. Open this form up and add a Form Load and a Form Closing event.

In the form closing event we will add 2 lines of code. These function calls will allow us to reset everything back to how it was when we close the shell.

```
Code Functions.RestoreDesktopArea()
```

```
Functions.ShowTaskBar()
```

In the form load event we will add this code. Just 4 lines to start us off. We have to function calls, one to hide the current task bar and another to create our new desktop working area. After this we need to display our shell.

```
Code Application.EnableVisualStyles()
```

```
Functions.HideTaskBar()
```

```
Functions.MakeNewDesktopArea()
```

```
BasicShell.Show()
```

Once we have all that done on to making the Shell.

Create a new form and call it whatever you like. There is a few things you will need to do to set this form up. In the form properties make these changes.

```
FormBorderStyle = None
```

```
Size = 600, 25
```

```
ShowInTaskbar = False
```

You will also need to add some controls to this form.

A Button call it whatever you like this will act as our start button.

A ComboBox this is going to be a our tab/window control. Call this 'windowList'

You will also need to ListBox's call one liveList and the other updateList. Make sure both of them are not visible on the form.

```
Visible = False.
```

You can add a label if you like, you can use to display the current time.

Last add a Timer control and Enable it.

Ok now for the code.

Imports. We need to make a few Imports first.

```
Code Imports BasicShell.Functions
```

```
Imports System.Runtime.InteropServices
```

```
Imports System.Text
```

In the form load event you will need to add this line of code.

What this does is set's our form position and makes sure we are in set location for the shell.

This all so extends our shell across the screen as well as setting the high. to the 25px we set on the form.

```
Code WinAPI.SetWindowPos(Me.Handle, 0, 0, 0, SystemInformation.VirtualScreen.Width, 25, &H40)
```

Create an event for windowList SelectedIndexChanged, then add this code to the event.

What we are doing here is allowing the user to be able to select a window and display it.

```
Code Dim windowName As String = windowList.Text
```

```
Dim handle As IntPtr = WinAPI.FindWindow(Nothing, windowName)
```

```
WinAPI.ShowWindow(handle, 1)
```

```
WinAPI.SetForegroundWindow(handle)
```

Now create an event for our timer Tick, then add this code here.

What this code does is updates the items in our windowList by checking if they are still open or if they have closed. By using the 2 listbox's we can avoid a constant update of the list. This way is will check before adding new items or removing old ones.

```
Code timeLbl.Text = TimeOfDay
```

```
Dim ar As ArrayList = GetActiveTasks()
```

```
updateList.Items.Clear()
```

```
liveList.Items.Clear()
```

```
For i As Integer = 0 To ar.Count - 1
```

```
Dim w As WindowData = DirectCast(ar(i), WindowData)
```

```
liveList.Items.Add(w.title)
```

```
If windowList.Items.Contains(w.title) Then
```

```
Else
```

```
    windowList.Items.Add(w.title)
```

```
End If
```

```
Next
```

```
For Each item In windowList.Items
```

```
updateList.Items.Add(item)
```

```
Next
```

```
For Each item In updateList.Items
```

```
If Not liveList.Items.Contains(item) Then
```

```
    windowList.Items.Remove(item)
```

```
End If
```

```
Next
```

That's basically it. If you can improve on it let me know. It's really messy and poorly coded.

It could do with removing the timer and using threading. It could also use a better method for handling the the window update. Like I said this was hacked together.

nuk - October 29, 2014, 6:57 am

Just curious... by "hacked together"; what you're really saying is Copy&Pasted?

Blixuk - October 29, 2014, 12:26 pm

No not at all. Yeah I read up some different stuff here and there. I converted a small tutorial I found from C# to VB. The tutorial only covered some of the concept and used a different method to display the windowList. I copied the WinAPI and also added some stuff. When I say hacked together I mean like taking one thing and changing it and adding stuff to get it to work how I want. There isn't anything really covered on this in VB. But I didn't just C&P something and post it here claiming I made it all. Like I said I'm sharing because maybe someone can use it and improve on it.

\$lick - October 29, 2014, 3:52 pm

(Oct 29, 2014 06:57 AM)nuk Wrote: [\[CITE\]](#) Just curious... by "hacked together"; what you're really saying is Copy&Pasted?

I can't find anything from google that's a direct copy/paste...

nuk - October 29, 2014, 7:50 pm

Okay that sweet then.... I was just wondering if that's what you meant is all ;) I shall check this out good sir and give a proper review :) Sorry if it seemed like I was accusing you of C/P ;)

Blixuk - *October 29, 2014, 8:47 pm*

It's perfectly fine. I know there is a lot of C&P on here. Thanks for taking the time to read it. I can add source or images if need be.