

"Error Creating Window Handle"

Asked 16 years, 3 months ago Modified 3 months ago

Viewed 80k times



37



We're working on a very large .NET WinForms composite application - not CAB, but a similar home grown framework. We're running in a Citrix and RDP environment running on Windows Server 2003.

We're starting to run into random and difficult to reproduct "Error creating window handle" error that seems to be an old fashion handle leak in our application. We're making heavy use of 3rd Party controls (Janus GridEX, Infralution VirtualTree, and .NET Magic docking) and we do a lot of dynamic loading and rendering of content based on metadata in our database.

There's a lot of info on Google about this error, but not a lot of solid guidance about how to avoid issues in this area.

Does the stackoverflow community have any good guidance for me for building handle-friendly winforms apps?

.net

winforms

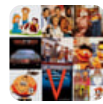
handles

Share

edited Feb 27, 2013 at 8:15

Improve this question

Follow



Kiquenet

15k ● 36 ● 151 ● 247

asked Sep 18, 2008 at 0:26



user8133

451 ● 1 ● 5 ● 5

-
- 1 See also [this post of mine about "Error creating window handle"] [1] and how it relates to USER Objects and the Desktop Heap. I provide some solutions. [1]:
weblogs.asp.net/fmarguerie/archive/2009/08/07/... – Fabrice
Aug 8, 2009 at 0:41
-

11 Answers

Sorted by:

Highest score (default)



I have tracked down a lot of issues with UIs not unloading as expected in WinForms.

32

Here are some general hints:



- alot of the time, a control will stay in use because controls events are not properly removed (the tooltip provider caused us really large issues here) or the controls are not properly Disposed.
- use 'using' blocks around all modal dialogs to ensure that they are Disposed
- there are some control properties that will force the creation of the window handle before it is necessary (for example setting the ReadOnly property of a TextBox control will force the control to be realized)

- use a tool like the [.Net Memory profiler](#) to get counts of the classes that are created. Newer versions of this tool will also track GDI and USER objects.
- try to minimize your use of Win API calls (or other DllImport calls). If you do need to use interop, try to wrap these calls in such a way that the using/Dispose pattern will work correctly.

Share Improve this answer

Follow

edited Oct 25, 2016 at 4:44



Jeremy Thompson

65.4k ● 37 ● 220 ● 338

answered Sep 18, 2008 at 0:44



Jack Bolding

3,829 ● 3 ● 42 ● 45



Understanding this error

14

Pushing the Limits of Windows: USER and GDI Objects – Part 1 by Mark Russinovich:



<https://blogs.technet.microsoft.com/markrussinovich/2010/02/24/pushing-the-limits-of-windows-user-and-gdi-objects-part-1/>

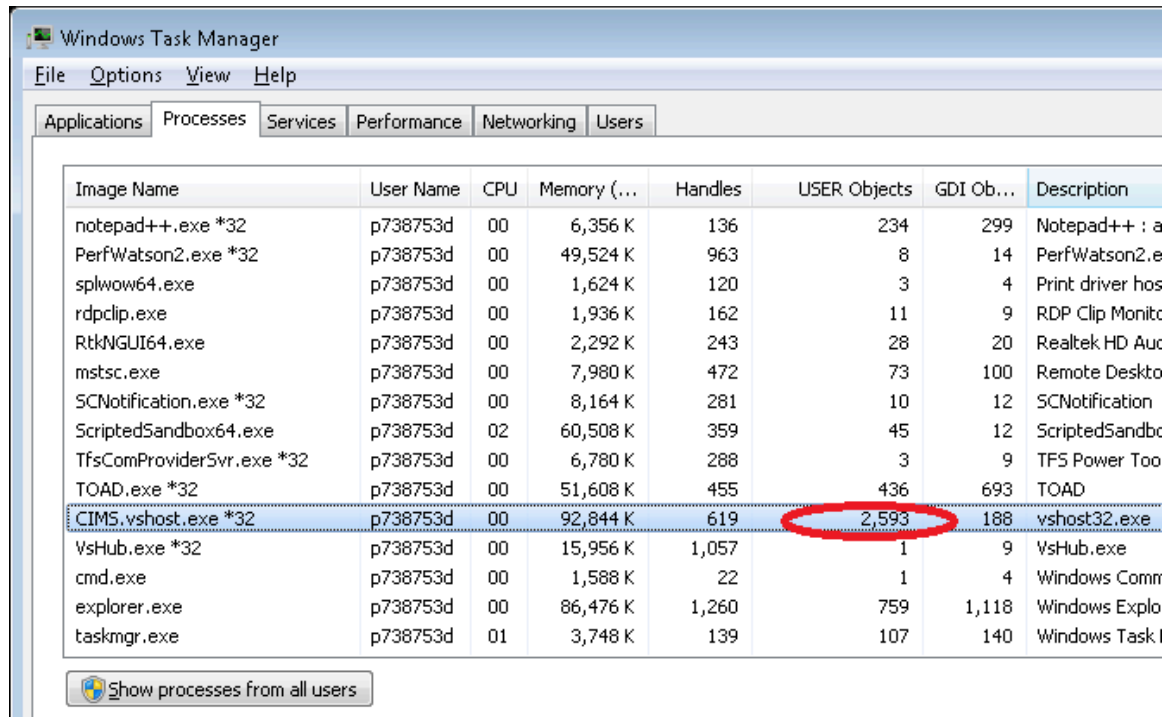


Troubleshooting this error

You need to be able to reproduce the problem. Here is one way of recording the steps to do that

<https://stackoverflow.com/a/30525957/495455>.

The easiest way to work out what is creating so many handles is to have TaskMgr.exe open. In TaskMgr.exe you need to have the USER Object, GDI Object and Handles columns visible as shown, to do this choose View Menu > Select Columns:



Windows Task Manager

File Options View Help

Applications Processes Services Performance Networking Users

Image Name	User Name	CPU	Memory (...)	Handles	USER Objects	GDI Ob...	Description
notepad++.exe *32	p738753d	00	6,356 K	136	234	299	Notepad++ : a
PerfWatson2.exe *32	p738753d	00	49,524 K	963	8	14	PerfWatson2.e
splwow64.exe	p738753d	00	1,624 K	120	3	4	Print driver hos
rdpclip.exe	p738753d	00	1,936 K	162	11	9	RDP Clip Monitc
RtkNGUI64.exe	p738753d	00	2,292 K	243	28	20	Realtek HD Auc
mstsc.exe	p738753d	00	7,980 K	472	73	100	Remote Deskto
SCNotification.exe *32	p738753d	00	8,164 K	281	10	12	SCNotification
ScriptedSandbox64.exe	p738753d	02	60,508 K	359	45	12	ScriptedSandbc
TfsComProviderSvr.exe *32	p738753d	00	6,780 K	288	3	9	TFS Power Too
TOAD.exe *32	p738753d	00	51,608 K	455	436	693	TOAD
CIMS.vshost.exe *32	p738753d	00	92,844 K	619	2,593	188	vshost32.exe
VsHub.exe *32	p738753d	00	15,956 K	1,057	1	9	VsHub.exe
cmd.exe	p738753d	00	1,588 K	22	1	4	Windows Commr
explorer.exe	p738753d	00	86,476 K	1,260	759	1,118	Windows Explo
taskmgr.exe	p738753d	01	3,748 K	139	107	140	Windows Task I

Show processes from all users

Go through the steps to cause the problem and watch the USER Object count increase to around 10,000 or GDI Objects or Handles reach their limits.

When you see the Object or Handles increase (typically dramatically) you can halt the code execution in Visual Studio by clicking the Pause button.

Then just hold down F10 or F11 to cruise through the code watching when the Object/Handle counts increases dramatically.

The best tool I have found so far is GDIView from NirSoft, it breaks up the GDI Handle fields:

Process ID	Process Name	Process Path	Process User	Pen	Brush	Bitmap	Font	Palette	Region	DC	All GDI
5520	OUTLOOK.EXE	C:\Program Files (x86)\...	SKYRMION\Alois	20	73	83	219	1	36	41	593
7480	WindowsLiveWrite...	C:\Program Files (x86)\...	SKYRMION\Alois	0	6	316	16	2	10	13	382
8768	notepad++.exe	C:\Program Files (x86)\...	SKYRMION\Alois	0	22	65	50	0	8	30	356
7864	devenv.exe	C:\Program Files (x86)\...	SKYRMION\Alois	24	33	91	13	1	12	61	262
1872	feedreader.exe	C:\Program Files (x86)\F...	SKYRMION\Alois	1	21	98	16	7	4	46	222

Handle	Object Type	Kernel Address	Extended Information	Detect Counter	Detected On
0xd5051250	Bitmap	0xffff90140706010		1	23.06.2016 22:36:43
0xd80a0648	Font	0xffff901423f0a10		1	23.06.2016 22:36:43
0xd80a13df	Font	0xffff90142242010		1	23.06.2016 22:36:43
0xd91017af	Brush	0xffff90142618560		1	23.06.2016 22:36:43
0xda050601	Bitmap	0xffff90144727ca0		1	23.06.2016 22:36:43

91 Processes, 1 Selected NirSoft Freeware, <http://www.nirsoft.net>

I tracked it down to this code used when setting DataGridViews "Filter Combobox" Columns Location and Width:

```

If Me.Controls.ContainsKey(comboName) Then
    cbo = CType(Me.Controls(comboName), ComboBox)
    With cbo
        .Location = New System.Drawing.Point(cumulativ
        .Width = Me.Columns(i).Width
    End With
    'Explicitly cleaning up fixed the issue of releasi
    cbo.Dispose()
    cbo = Nothing
End If

```

In my case (above) the solution was **explicitly disposing** and cleaning up that fixed the issue of releasing USER objects.

This is the stack trace:

```

at
System.Windows.Forms.Control.CreateHandle()
at
System.Windows.Forms.ComboBox.CreateHandl
le() at
System.Windows.Forms.Control.get_Handle() at

```

System.Windows.Forms.ComboBox.InvalidateEverything() at
System.Windows.Forms.ComboBox.OnResize(EventArgs e) at
System.Windows.Forms.Control.OnSizeChanged(EventArgs e) at
System.Windows.Forms.Control.UpdateBounds(Int32 x, Int32 y, Int32 width, Int32 height, Int32 clientWidth, Int32 clientHeight) at
System.Windows.Forms.Control.UpdateBounds(Int32 x, Int32 y, Int32 width, Int32 height) at
System.Windows.Forms.Control.SetBoundsCore(Int32 x, Int32 y, Int32 width, Int32 height, BoundsSpecified specified) at
System.Windows.Forms.ComboBox.SetBoundsCore(Int32 x, Int32 y, Int32 width, Int32 height, BoundsSpecified specified) at
System.Windows.Forms.Control.SetBounds(Int32 x, Int32 y, Int32 width, Int32 height, BoundsSpecified specified) at
System.Windows.Forms.Control.set_Width(Int32 value)

Here is the crux of [a helpful article by Fabrice](#) that helped me work out the limits:

"Error creating window handle"

When a big Windows Forms application I'm working on for a client is used actively, users often get "Error creating window handle" exceptions.

Aside from the fact that the application consumes too much resources, which is a separate issue altogether that we are already addressing, we had difficulties with determining what resources were getting exhausted as well as what the limits are for these resources. We first thought about keeping an eye on the Handles counter in the Windows Task Manager. That was because we noticed that some processes tended to consume more of these resources than they normally should. However, this counter is not the good one because it keeps track of resources such as files, sockets, processes and threads. These resources are named Kernel Objects.

The other kinds of resources that we should keep an eye on are the GDI Objects and the User Objects. You can get an overview of the three categories of resources on MSDN.

User Objects

Window creation issues are directly related to User Objects.

We tried to determine what the limit is in terms of User Objects an application can use. There is a quota of 10,000 user handles per process. This value can be changed in the registry, however this limit was not the real show-stopper in our case. The other limit is 66,536 user handles per Windows session. This limit is theoretical. In practice, you'll notice that it can't be reached. In our case, we were getting the dreaded "Error creating window

handle" exception before the total number of User Objects in the current session reached 11,000.

Desktop Heap

We then discovered which limit was the real culprit: it was the "Desktop Heap". By default, all the graphical applications of an interactive user session execute in what is named a "desktop". The resources allocated to such a desktop are limited (but configurable).

Note: User Objects are what consumes most of the Desktop Heap's memory space. This includes windows. For more information about the Desktop Heap, you can refer to the very good articles published on the NTDebugging MSDN blog:

What's the real solution? Be green!

Increasing the Desktop Heap is an effective solution, but that's not the ultimate one. The real solution is to consume less resources (less window handles in our case). I can guess how disappointed you can be with this solution. Is this really all what I can come up with?? Well, there is no big secret here. The only way out is to be lean. Having less complicated UIs is a good start. It's good for resources, it's good for usability too. The next step is to avoid waste, to preserve resources, and to recycle them!

Here is how we're doing this in my client's application:

We use TabControls and we create the content of each tab on the fly, when it becomes visible; We use expandable/collapsible regions, and again fill them with

controls and data only when needed; We release resources as soon as possible (using the Dispose method). When a region is collapsed, it's possible to clear it's child controls. The same for a tab when it becomes hidden; We use the MVP design pattern, which helps in making the above possible because it separates data from views; We use layout engines, the standard FlowLayoutPanel and TableLayoutPanel ones, or custom ones, instead of creating deep hierarchies of nested panels, GroupBoxes and Splitters (an empty splitter itself consumes three window handles...). The above are just hints at what you can do if you need to build rich Windows Forms screens. There's not doubt that you can find other approaches. The first thing you should do in my opinion is building your applications around use cases and scenarios. This helps in displaying only what's needed at a given time, and for a given user.

Of course, another solution would be to use a system that doesn't rely on handles... WPF anyone?

[Share](#) [Improve this answer](#)

[edited Jul 6, 2021 at 8:25](#)

[Follow](#)

answered Oct 25, 2016 at 5:30



[Jeremy Thompson](#)

65.4k ● 37 ● 220 ● 338



I had this error when I subclassed NativeWindow and called CreateHandler manually. The problem was I forgot

9

to add `base.WndProc(m)` in my overridden version of `WndProc`. It caused the same error



Share Improve this answer

answered Dec 10, 2016 at 21:22

Follow



[aderesh](#)

947 ● 11 ● 23

-
- 1 I had a similar bout of forgetfulness and this solution reminded me to go back and check. Problem solved. Thanks.
– [Yonabart](#) Mar 17, 2017 at 10:03

@Yonabart, glad to help:) – [aderesh](#) Oct 12, 2018 at 19:43

I had modified an overridden version of `WindProc` to not execute `base.WndProc(m)` on loading and this caused the problem – [SimonKravis](#) Jul 30, 2020 at 9:34



5

I met this exception because endless loop creating new UI control and set its properties. After looped many times, this excption was thrown when change control visible property. I found both User Object and GDI Object (From Task Manager) are quite large.



I guess your issue is similar reason that system resources are exhaust by those UI controls.

Share Improve this answer

answered Jan 13, 2012 at 7:12

Follow



[sliu](#)

51 ● 1 ● 1



4



I am using the Janus Controls at work. They are extremely buggy as far as disposing of themselves go. I would recommend that you make sure that they are getting disposed of correctly. Also, the binding with them sometimes doesn't release, so you have to manually unbind the object to dispose of the control.

Share Improve this answer

answered Sep 18, 2008 at 0:47

Follow



MagicKat

9,811 ● 7 ● 34 ● 43



3



I faced this exception while adding controls in to the panel, Because in panel child controls not cleared. If dispose the child controls in panel then bug fixed.

```
For k = 1 To Panel.Controls.Count  
    Panel.Controls.Item(0).Dispose()  
Next
```

Share Improve this answer

edited Sep 3, 2016 at 5:55

Follow



Pang

10.1k ● 146 ● 85 ● 124

answered Sep 3, 2016 at 5:51



Sudhakar Mallu

31 ● 1

Thanks, I had the same situation. I had a panel with multiple controls inside. – CABascourt Jan 28, 2019 at 18:34



1

In my case I was overriding `WndProc(ref Message m)` but not calling `base.WndProc(ref m);`

Share Improve this answer

answered Sep 23, 2020 at 14:56

Follow



Vapid

816 ● 8 ● 30



1

Greeting, I facing same problem due to large amount of PictureBox for image processing was added to panel. Original code use `Controls.RemoveAt(0)` still cause the problem. Just changed to `Controls[0].Dispose ->` seem like problem away.



```
while (mGridPanel.Controls.Count > 0)
    mGridPanel.Controls[0].Dispose();
```

Share Improve this answer

answered Oct 9, 2022 at 22:59

Follow



YUT

21 ● 4

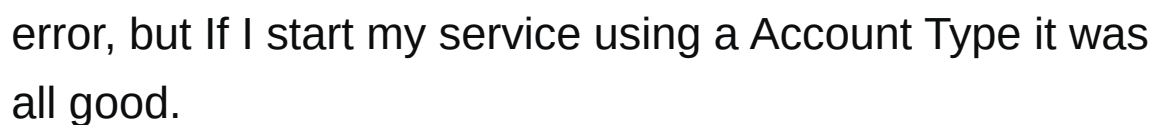


0

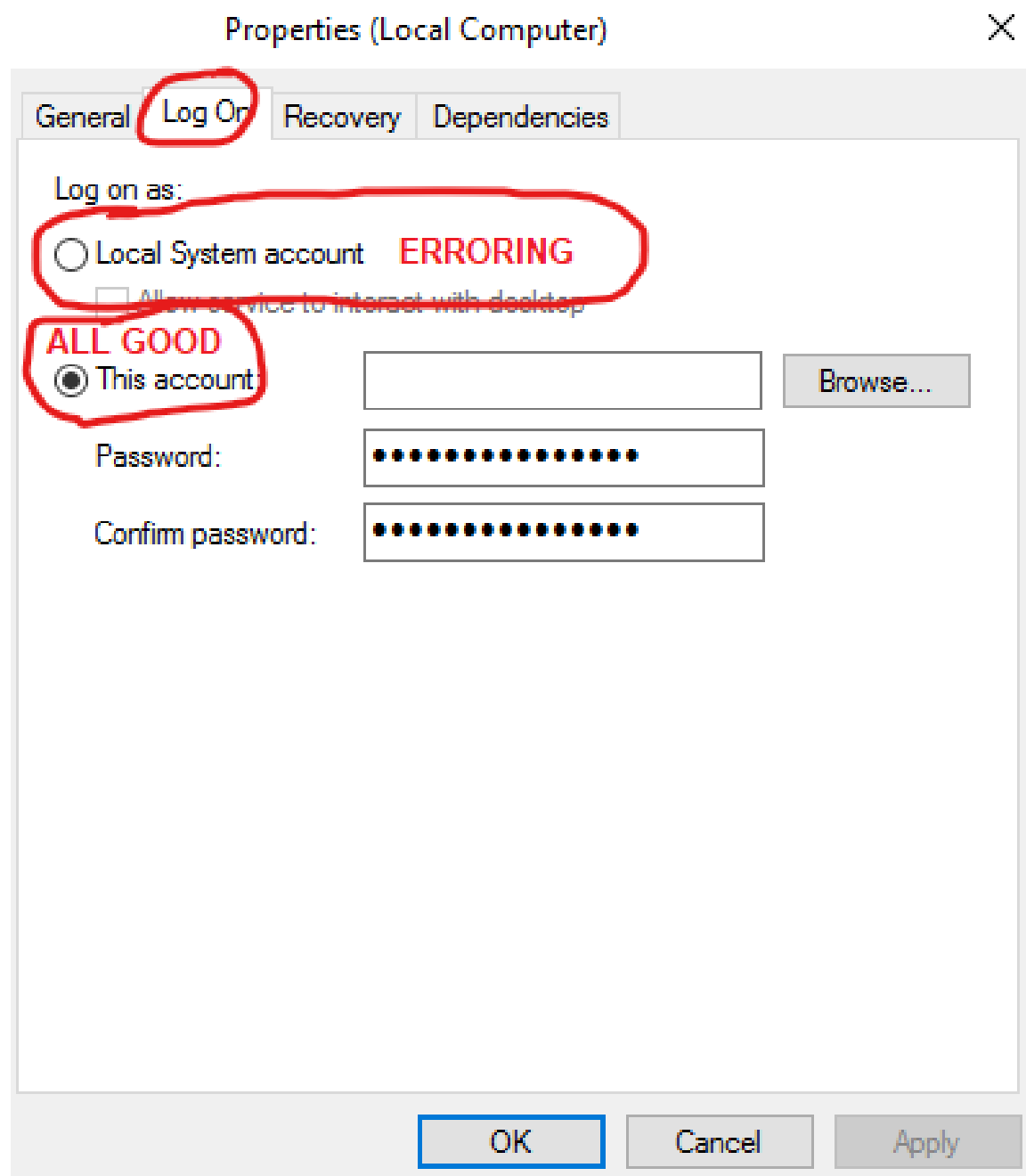
I had the same error but in my C# Windows Service. Inside my service, in `OnStart()` method I was creating a new WinForms Object using a "using" statement. Just after that my service was throwing this error.



My fix was simple, it turned out that if I start my service using a Log On As "Local System" it was throwing that



I hope someone will find this useful.



Share Improve this answer

answered Apr 4, 2022 at 17:47

Follow



alaneo

89 • 4



I ran into the same .Net runtime error but my solution was different.

-1



My Scenario: From a popup Dialog that returned a DialogResult, the user would click a button to send an email message. I added a thread so the UI didn't lock up while generating the report in the background. This scenario ended up getting that unusual error message.



The code that resulted in problem: The problem with this code is that the thread immediately starts and returns which results in the DialogResult being returned which disposes the dialog before the thread can properly grab the values from the fields.

```
private void Dialog_SendEmailSummary_Button_Click(object
{
    SendSummaryEmail();
    DialogResult = DialogResult.OK;
}

private void SendSummaryEmail()
{
    var t = new Thread(() => SendSummaryThread(Textbox
Textbox_Body.Text, Checkbox_IncludeDetails.Checked));
    t.Start();
}

private void SendSummaryThread(string subject, string
includeTestNames)
{
    // ... Create and send the email.
}
```

The fix for this scenario: The fix is to grab and store the values before passing them into the method which creates the thread.

```
private void Dialog_SendEmailSummary_Button_Click(object sender, EventArgs e)
{
    SendSummaryEmail(Textbox_Subject.Text, Textbox_Body.Text, Textbox_IncludeDetails.Checked);
    DialogResult = DialogResult.OK;
}

private void SendSummaryEmail(string subject, string body, bool includeTestNames)
{
    var t = new Thread(() => SendSummaryThread(subject, body, includeTestNames));
    t.Start();
}

private void SendSummaryThread(string subject, string body, bool includeTestNames)
{
    // ... Create and send the email.
}
```

Share Improve this answer

answered Sep 23, 2016 at 20:58

Follow



GrayDwarf

2,725 ● 2 ● 23 ● 24



-1



same error occurred when i started using threading in my WinForm App, i used stack trace to find what is throwing error and found out UltraDesktopAlert component of Infragistics was behind this so i invoked it differently and error is now gone.



```
this.Invoke((MethodInvoker)delegate
{
    //call your method here
});
```

the full code will look like this.

```
private void ultraButton1_Click(object sender, EventArgs e)
{
    Task.Factory.StartNew(() => myMethod1());
}

void myMethod1()
{
    //my logic

    this.Invoke((MethodInvoker)delegate
    {
        ultraDesktopAlert1.Show($"my message header",
        });

    //my logic
}
```

also i was unable to use GDI utility to find how many handle my app creates but my app (64bit) was not available in its list. another solution was to change desktop heap value to `SharedSection=1024,20480,768` at following location HKEY

```
Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SubSystems
```

but mine was already with same values. only invoke method delegate worked for me. hope this helped.

Share Improve this answer

answered May 1, 2019 at 18:36

Follow



Abubakar Riaz

340 ● 9 ● 27
