

Forum:[Hack Forums](#)

Thread: [Ultimate Flashing Text](#)

RandomUser123 - December 27, 2014, 2:14 pm

Well, i saw few threads trying to make a label flashing, now i made mine one to help people using timers learning to use threads.

PHP Code

```
<?php
```

```
Imports System.Threading
```

```
Module Effects
```

```
    Public Function FlashingLabelAsync(ByVal Label As Label, ByVal Colors As List(Of Color),  
ByVal Interval As Integer) As Thread  
        Dim Thread1 As New Thread(Sub()  
            Try  
                While True  
                    For Each CurrentColor As Color In Colors  
                        Label.Invoke(Sub() Label.ForeColor = CurrentColor)  
                        Thread.Sleep(Interval)  
                    Next  
                End While  
            Catch  
                Thread.CurrentThread.Abort()  
            End Try  
        End Sub)  
        Return Thread1  
    End Function
```

```
End Module
```

```
Imports System.Threading
```

```
It's needed to use "Thread"
```

```
Public Function
```

```
Its a method which returns something,
```

```
Public Function Addition() As Integer 'As Integer is the type of the returned data
```

```
Return 2+2
```

```
End Function
```

```
MsgBox(Addition) 'This will show up a message box with written "4" (2+2)
```

```
ByVal Label As Label
```

```
ByVal means if function can modify or not "Label".
```

```
ByRef allows to edit it, ByVal doesn't.
```

```
As Thread
```

```
Is the type of returned data, this function will return a thread so using "FlashingLabelAsync" is like
```

creating a new thread, but it's content is precoded to make a label flash.

`Dim Thread1 As New Thread(Sub() ... )`

It's to declare a new thread called Thread1, Sub() (followed by code to execute) is the method that this thread will run. Using a thread to execute a method will not occupy the main process, so application will not freeze while executing it.

Methods running code in another thread are called "Async".

`Try ... Catch ... End Try`

Tries to execute the code between "Try" and "Catch". If during execution it gets an error, will run the code between "Catch" and "End Try"

`While True`

`...`

`End While`

While [condition] executes the code between While ... End While for all the time which the condition will be true, when condition will be false, it will stop executing code.

It's needed to run the code forever in the thread, at least until thread gets closed.

`For Each CurrentColor As Color In Colors`

`...`

`Next`

Gets all items from a list/array and processes them 1 per time, giving you possibility to use a variable (CurrentColor As Color) which will be one of the items in "Colors". Then process the next item.

`Label.Invoke(Sub() ... )`

As for threads it executes code next to "Sub()" but it inherits the label declared in the function.

This allows to edit GUI (Graphic User Interface) from the thread to the process.

`Thread.Sleep(...)`

Replace ... with time (in milliseconds) you want the thread to wait when this code gets executed (1000 is a second).

`Thread.CurrentThread.Abort`

This code is only executed if an error occurred (like if main process gets closed without closing all threads before), if so this thread will stop executing and the label will stop flashing (and remain at current color).

`Return Thread1`

As in the previous example, Addition returned "4" (2+2), this function Returns the just created Thread.

How to use it?

PHP Code

`<?php`

`Dim FlashingLabel1 As Threading.Thread = FlashingLabelAsync(Label1, Colors, 1000)`

`Dim Colors As New List(Of Color)`

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Colors.Add(Color.Red)
    Colors.Add(Color.Blue)
```

```
Colors.Add(Color.Green)
FlashingLabel1.Start()
End Sub
```

Dim Colors As New List(Of Color)

Declare a new variable (Called Colors) which is a list of color(s).

```
Colors.Add(Color.Red)
```

Add to the list "Colors" the value "Color.Red". The value between parenthesis must be the type defined in List(Of ...)

```
Dim FlashingLabel1 As Threading.Thread = FlashingLabelAsync(Label1, Colors, 1000)
```

Declare a new variable (a thread) which gets value from "FlashingLabelAsync", because FlashingLabelAsync returns the just coded thread.

Now define all variables needed to the method "FlashingLabelAsync" created, like "ByVal Label As Label" is the label which will be flashing. Set the list of colors as second variable, and the interval time to wait before to change color to the next one.

```
FlashingLabel1.Start()
```

It starts the thread returned by FlashingLabelAsync.

```
FlashingLabel1.Abort()
```

It's to stop thread which edits forecolor of the label.

If you declare the variable FlashingLabel1 in a method, you can't use/start/stop it outside the method. If you want to do so (like starting using a button and stopping using another one) you have to declare the variable "FlashingLabel1" in the class (Below Public Class Form1)

```
.Suspend() .Resume()
```

An aborted thread can't be restarted, so to temporarily stop it, you can use "Suspend" and "Resume" instead "Abort" and "Start".

You can handle Start & Resume in a single method (Like a button_click event) by declaring a variable as boolean and set it to true when you start the thread, and set it to false when you stop it. Then just check if that variable is true (thread running) or not.

+Haze - December 27, 2014, 2:38 pm

You should never abort a thread. You should always let it finish. I don't get why you declared a function to return a thread...

RandomUser123 - December 27, 2014, 2:44 pm

(Dec 27, 2014 03:38 PM)+Haze Wrote: [CITE](#) You should never abort a thread. You should always let it finish. I don't get why you declared a function to return a thread...

So you can make as many flashing label as you want by using a function instead creating everytime a new thread...

And While True makes the thread never stop, so it must be aborted.

Dialog - December 27, 2014, 3:27 pm

One of the golden rules of threading is that you should not access UI elements from threads other than the one they were created on, unless very necessary.

Timer's should be used to complete repetetive tasks which are not CPU intensive at an interval; therefore, those would be the best option here.

(Dec 27, 2014 03:44 PM)RandomUser123 Wrote: [\[CITE\]](#) So you can make as many flashing label as you want by using a function instead creating everytime a new thread...

And While True makes the thread never stop, so it must be aborted.+Haze is correct; it's bad practice to explicitly call Abort on a thread. Instead, consider setting a flag which is checked on the thread. That way, it has a chance to finish peacefully.

Although it is in C#, MSDN does a better job at explaining this than I can:
<http://msdn.microsoft.com/en-us/library/7a2f3ay4%28v=vs.90%29.aspx>

Thanks for taking the time to write this up nonetheless, good contribution apart from the things listed above.

RandomUser123 - December 27, 2014, 3:53 pm

(Dec 27, 2014 04:27 PM)Dialog Wrote: [\[CITE\]](#) One of the golden rules of threading is that you should not access UI elements from threads other than the one they were created on, unless very necessary.

Timer's should be used to complete repetetive tasks which are not CPU intensive at an interval; therefore, those would be the best option here.

+Haze is correct; it's bad practice to explicitly call Abort on a thread. Instead, consider setting a flag which is checked on the thread. That way, it has a chance to finish peacefully.

Although it is in C#, MSDN does a better job at explaining this than I can:
<http://msdn.microsoft.com/en-us/library/7a2f3ay4%28v=vs.90%29.aspx>

Thanks for taking the time to write this up nonetheless, good contribution apart from the things listed above.

Thanks for the clear explaining reply.

Hawk HF - December 27, 2014, 4:42 pm

I've tried to respond to your PMs, but it seems that you have PMs disabled.

RandomUser123 - December 27, 2014, 5:13 pm

(Dec 27, 2014 05:42 PM)Hawk HF Wrote: [\[CITE\]](#) I've tried to respond to your PMs, but it seems that you have PMs disabled.

Yea, sorry, i just activated them... My error :P