

GSMsgBox: Custom Message Box

If you use .NET to write Windows® apps, you well know that the .NET WinForms MessageBox is a handy way to show messages to the user easily.

Still, .NET's MessageBox, as handy as it is, was missing a few things I would like to have. As a result I put together something that is similar to .NET's MessageBox but addressing my specific needs.

I wanted to:

- tell the message box where to appear on the screen or to simply center itself to a WinForm.
- be able to colorize and format my message and possibly display an image.
- be able to use my own icon instead of what is available in the standard .NET MessageBox.
- be able to code using my message box like I did with the standard .NET Message Box.

I wrapped together standard WinForms controls around a web browser control and packaged it so I can re-use it in a .NET Assembly. After using it a few years I got to tweaking it so it is just right.

(For me at least. 😊)

I believe you will benefit by using this Message Box system. Something like this IS easy to make for any developer but it often takes time to refine so as to be the most useful. So I offer it as a free download so you can simply save yourself the time of making and stabilizing your own Message Box assembly application.

What is in the GSMsgBox assembly

There are 2 classes exposed.

TimedMessage class.

The TimedMessage Class is used like a message box. This class has a timer that closes the message automatically. I use the TimedMessage class occasionally when the applicaton needs to tell the user something and does not need the user to respond.

The MsgBox class


This is the class that can be used in place of the standard WinForms MessageBox.

How to use the classes

The ShowMsg or Show methods are the two I have most often used.

Consider the VB code snapshow below where I first use .NET's MessageBox and then GSMsgBox

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    MessageBox.Show("I am the Message", "I am the caption", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
    MsgBox.ShowMsg("I am the Message", "I am the caption", MessageBoxButtons.YesNo, GSNMsgBoxIcons.Exclamation, Me)
End Sub
```



Looks almost identical. Yet the last parameter in the example above makes all the difference.

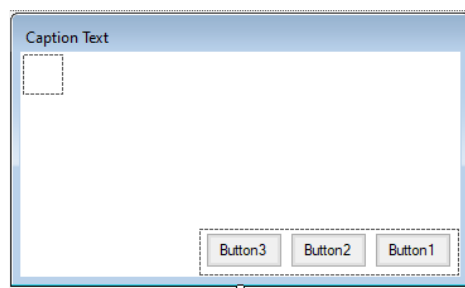
GMsgBox will appear centered in the form (Me/this) the button is contained in.

Here is the official signature of the ShowMsg method as seen from Visual Studio's Intellisense pop up.

```
Function MsgBox.ShowMsg(sMsg As String, Title As String, eButtons As MessageBoxButtons, elcon As GMsgBoxIcons, [HostForm As Form = Nothing], [DisplayHeight As Integer = -1], [DisplayWidth As Integer = -1], [DisplayTop As Integer = -1], [DisplayLeft As Integer = -1]) As DialogResult
```

The additional parameters are used to let you control the size and placement of the message box.

Inside the GMsgBox assembly I created a WinForm form with a panel of buttons, an image control to show the icon, and a WebBrowser control for the message of the text box. Then, I expose a MsgBox class that does all the work of manipulating the controls to emulate .NET's MessageBox in a method named Show. It is that simple. Below is a screenshot of the design mode of my MsgBox form.



MsgBox WinForm Controls

Having my own form to control, I am capable of setting the sizing and placing the Message Box form on the screen. Because the Message Box "message" is displayed in a web browser control, I planned to use HTML to provide formatting and coloring of the message. (*I could also show an image as well*)

I suggest you use the object browser to explore all the properties, methods, and enumerations. This is not a large assembly and quite simple in intent. I share more insight about the classes at the end of this document.

An additional tool

After using GMsgBox consistently over a few projects, I decided to make a utility WinForm application called "GMsgBox Setup" to allow me to better design my application's pop up Message Boxes.

As an output, the GMsgBox Setup application would generate the program code I would need in the application I was working in at the time.

The GMsgBox Setup application turned out to be a real benefit because I could do all of the Message Box design and visualization outside of the application and plug it in when I'm done. Because of this tool I did not have to step through my code using the debugger AND make sure the business logic associated to trigger the MsgBox inside of my application was set up, just so I could test the design of the MsgBox message and sizing.

The GMsgBox Setup utility is installed when you install the GMsgBox Assembly using the installer from my web site. (<http://www.gen-e-sys.com>)

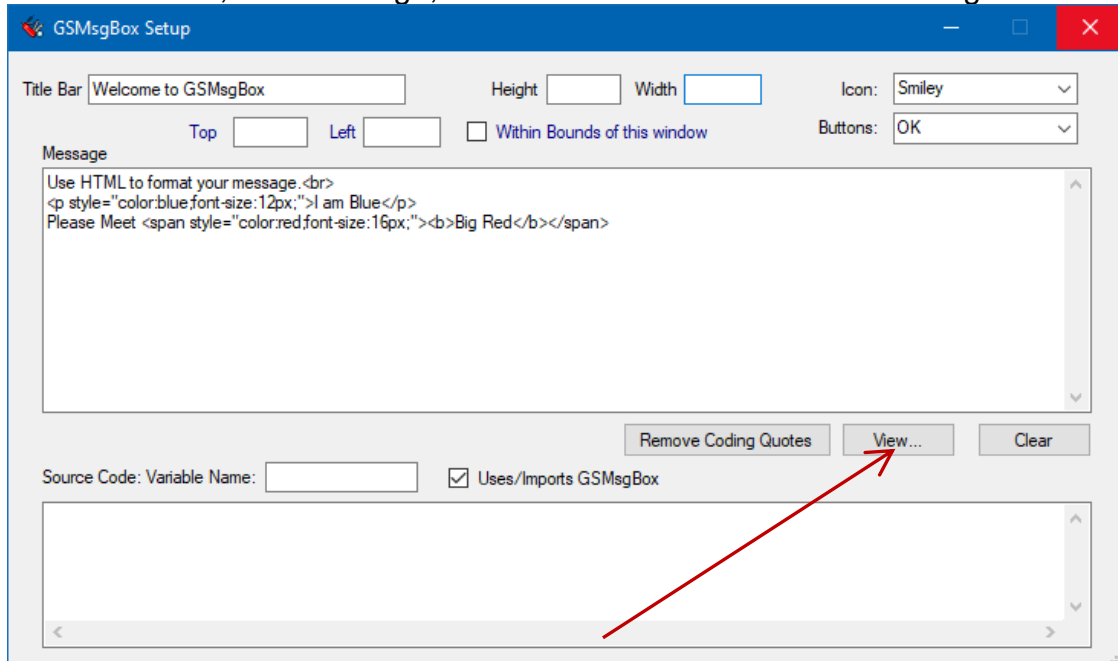
The following pages outline the steps of a single editing session using the GMsgBox Setup tool.

You will do his many times. (and enjoy it).

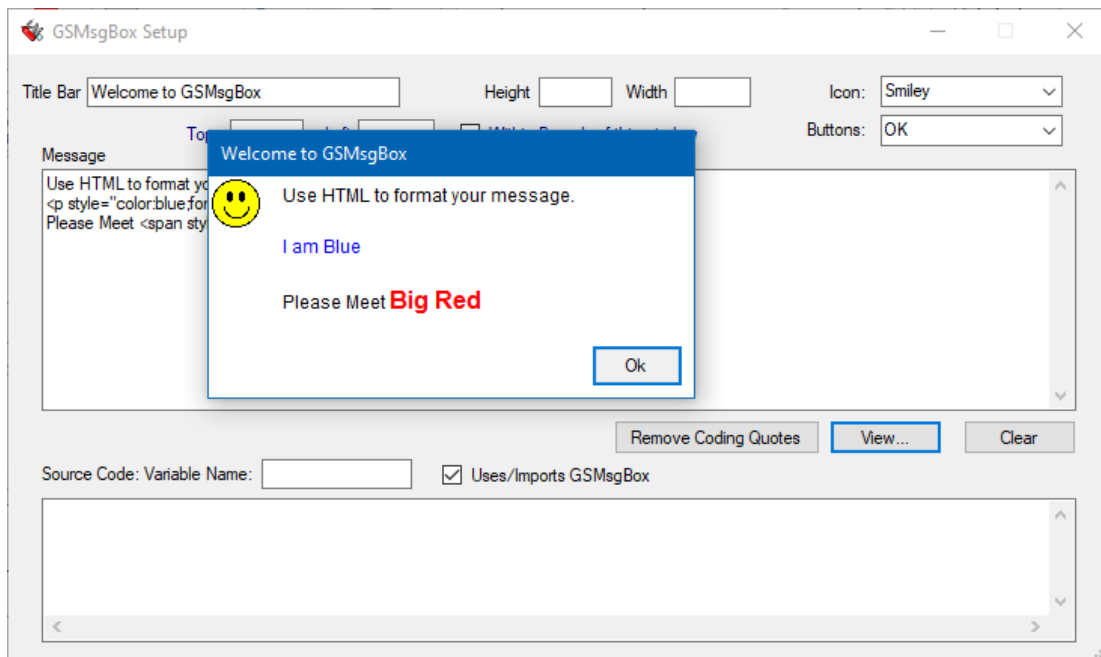
Creating a program statement using the GSMsgBox Setup Utility App

Start the MsgBoxSetup.exe utility application and the Main window shows (below).

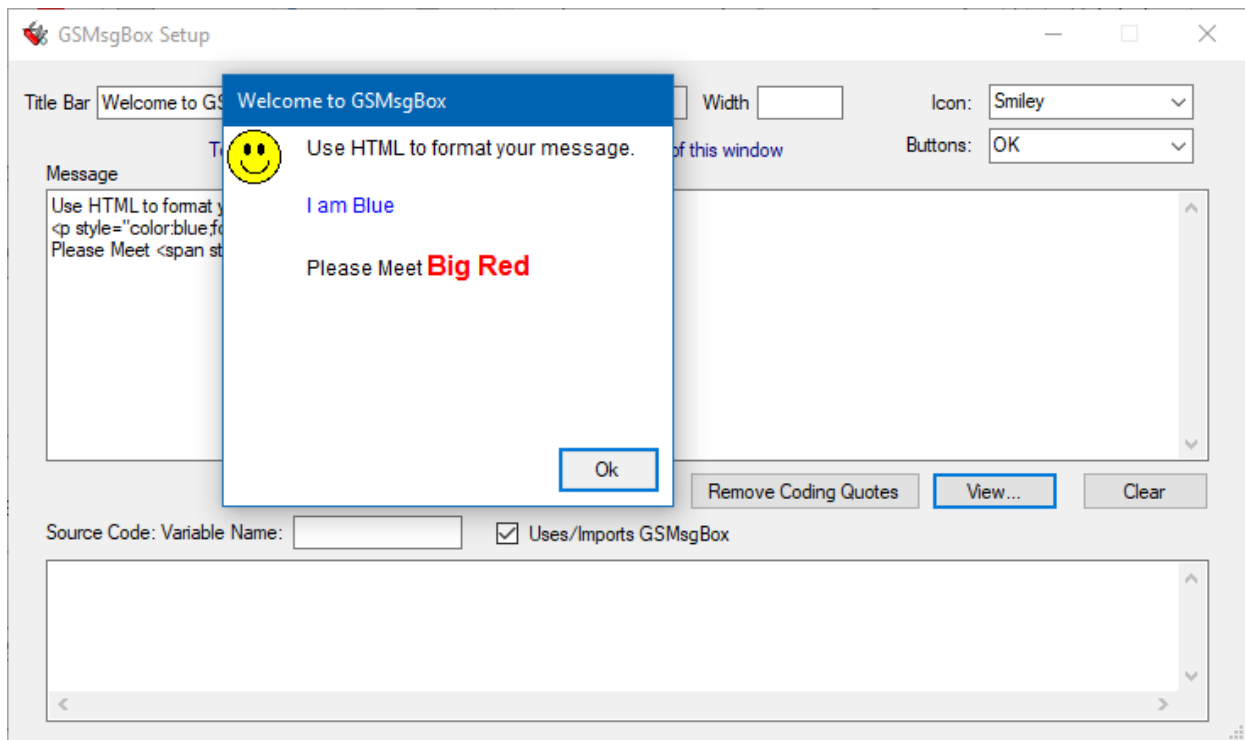
Enter a Title, and Message, then select an Icon and Button Configuration.



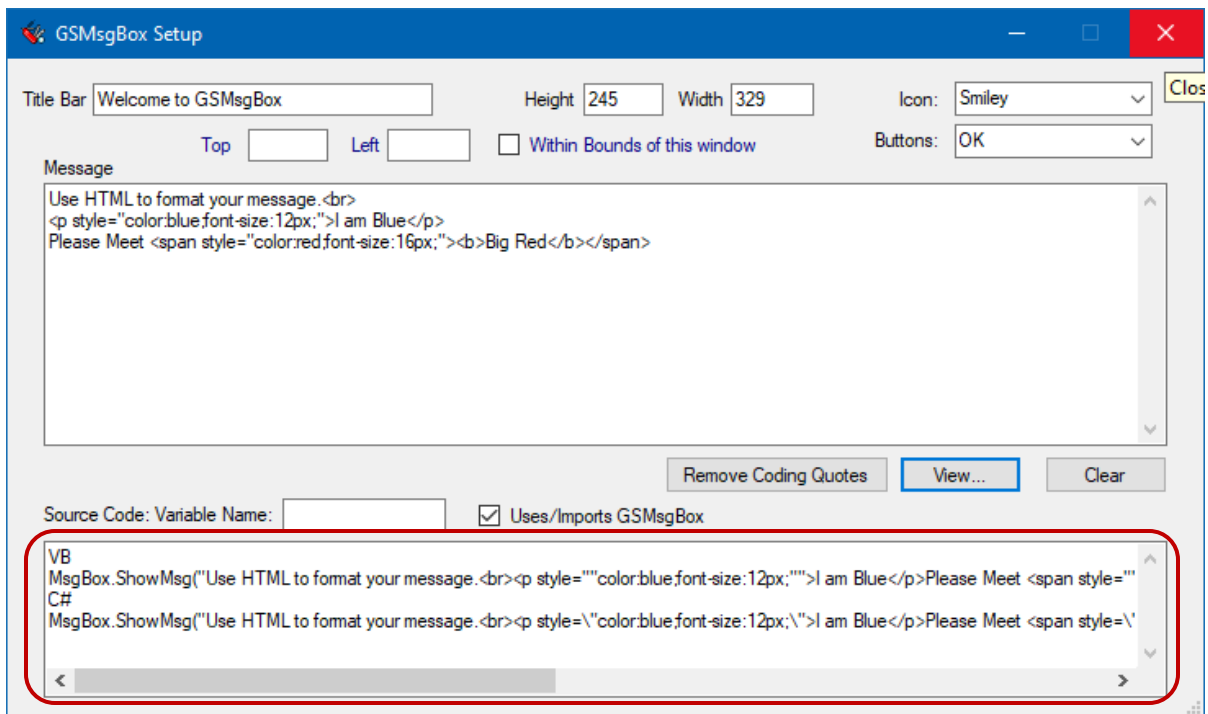
Then click the View button to visualize your message box.



Then size the Message Box with your mouse by dragging on the edges of the message box window.
(results shown below)....



After closing the re-sized message box, the source code and property text boxes are populated with the values of the window sizes as seen below.



More Importantly: Notice the source code text area on the bottom of the window. This is the output of the GSMsgBox Setup. The source code statement in VB and C# needed to show the user message box you just designed.

I just copy the line I want into the computer clipboard, then paste it into the application code I am working on. This tool makes creating polished user messages for Win32 .NET applications simple, convenient, and fun.

Assembly History/Author's Notes

Using the 1st version of this assembly, I had to set the individual Height and Width properties and then show the MsgBox in separate statements. (Sample below)

```
oMsgBox = New GSMsgBox.MsgBox
oMsgBox.Height = 100
oMsgBox.Width = 200
oMsgBox.Show("Size Smaller than Minimum", "Sorry", MessageBoxButtons.YesNoCancel, GSMsgBox.GSNMsgBoxIcons.Halt, Me)
```

The optional Form reference at the end of the Show method signature, (*shown above as Me or this in C#*), was key to GSMsgBox showing itself within the bounds of the Form presenting it. It worked great !!!!! I was pleased but...

As you can imagine, I got tired of multiple statements quickly.

In the next version (2.x), I appended optional Height, Width, Top, and Left values to the Show method signature. The result was very close to the way we use the "normal" .NET MessageBox, AND... provided me the control to size and place the location of the message box as well as colorize and format the text. In this version I thought I reached my goals I had when I started this effort.

Code Example of typical statement in VB after version 2.x.

```
MsgBox.Show("Size Smaller than Minimum", "Sorry", MessageBoxButtons.YesNoCancel, GSMsgBox.GSNMsgBoxIcons.Halt, Me, 100, 200)
```

or using names to explain the added optional parameters.

```
MsgBox.Show("Message", "Caption", "button config", "selected icon", Optional [ FormRef, height, width, top, left ])
```

Using the MsgBox or TimedMessage in one statement worked well and was in play for a number of years.

The current version of this assembly (3.x), added Shared/Static ShowMsg methods so you can use the assembly resources without spinning up an instance. (Looking back, I say I should have done *that* from the start.)

Not much more to say other than I love this cute little assembly. It works. Beautifully.