\_\_\_\_\_ \_ \_ \_ \_\_\_\_\_\_ \_

/ \_\_\_\_| | | (\_) | \_\_\_\_| | |

| (\_\_\_ | |\_ \_ \_| |\_ \_ \_\_ \_\_ \_| |\_\_ \_ \_\_ \_\_ \_ \_ \_\_ \_\_\_ \_\_\_\_\_ \_\_\_\_\_ \_ \_\_| | \_\_

\\_\_\_ \| \_\_| | | | | | '\_ \ / \_` | \_\_| '\_\_/ \_` | '\_ ` \_ \ / \_ \ \ /\ / / \_ \| '\_\_| |/ /

\_\_\_\_) | |\_| |\_| | | | | | | (\_| | | | | | (\_| | | | | | | \_\_/\ V V / (\_) | | | <

|\_\_\_\_\_/ \\_\_|\\_\_, |\_|\_|\_| |\_|\\_\_, |\_| |\_| \\_\_,\_|\_| |\_| |\_|\\_\_\_| \\_/\\_/ \\_\_\_/|\_| |\_|\\_\

\_\_/ | \_\_/ |

|\_\_\_/ |\_\_\_/

**The beginning**

Programming with MFC and the native MS-Windows resources has a somewhat limiting possibilities, and always leaves you with a program that has a sort of outdated look-and-feel. A look-and-feel that is mostly described as the dull gray windows interface.

That’s the reason that some day a few years ago I decided to have a go at my own custom-made interface.

Well, why not buy it somewhere? There are propriety frameworks like CodeJock ([www.codejock.com](http://www.codejock.com)) or the bgc interface ([www.bgc.com](http://www.bgc.com)). But then again: I would not be able to use it in some of the opensource programs that I made. And that has a somewhat limiting effect. So, I wanted a framework that was open-source, but came away from the standard gray MS-Windows interface.

Two elements stand out, that came first. These were the colored scroll bars and the colored non-client area of a program. In order to be able to paint the non-client area, we need to handle all the non-client area windows messages. Beginning with WM\_NCCALCSIZE and WM\_NCPAINT to all non-client mouse messages.

The second important element where the scrollbars. The idea and the basic code came from this codeproject article:

[Replace a Window's Internal Scrollbar with a customdraw scrollbar Control](https://www.codeproject.com/Articles/14724/Replace-a-Window-s-Internal-Scrollbar-with-a-custo).

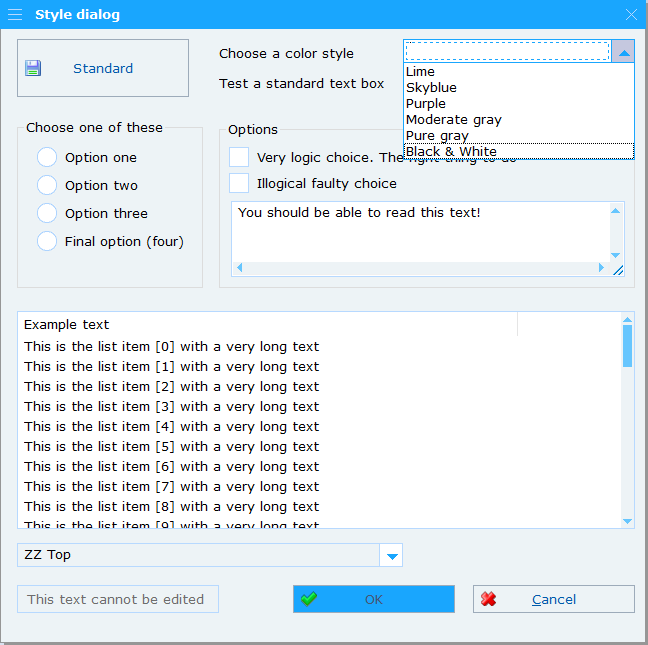
**The basics**

One of the main concerns was that it should be possible to convert rather large code bases to the new styling. It would be awkward to change the style, and then having to retest the whole application, just because it looked differently. The way to do this was to have a one-to-one translation between the native MFC controls that I used all the time, and the new styling controls. This translation is best viewed in the “StyleFramework.h” header, but is also shown in the table below:

|  |  |
| --- | --- |
| Standard MFC control or object | Style framework object |
| CDialog | StyleDialog |
| CFormView | StyleFormView |
| CFrameWnd | StyleFrameWnd |
| CFrameWndEx | StyleFrameWndEx |
| AfxMessageBox | StyleMessageBox |
| CMDIFrameWndEx | StyleMDIFrameWnd |
| CButton | StyleButton |
| CCalendarCtrl | StyleCalendarCtrl |
| CButton | StyleCheckBox / StyleRadioButton |
| CComboBox | StyleComboBox |
| CEdit | StyleEdit |
| CGridCtrl | StyleGridCtrl |
| CLinkCtrl | StyleHyperlinkCtrl |
| CListBox | StyleListBox |
| CListCtrl | StyleListCtrl |
| CProgressCtrl | StyleProgressCtrl |
| CStatic | StyleStatic |
| CSpinButtonCtrl | StyleButtonCtrl |
| CTabCtrl | StyleTabCtrl |
| CTreeCtrl | StyleTreeCtrl |

Seen from this table the general conversion instruction seems simple: just replace the Cxxxxx control with the Stylexxxxx counterpart and voilá. And well….. 95 percent is indeed that simple!

This is an example from the “TestDialogs” project, showing the “skyblue” theme (there are 6 color themes). What stands out is that the default font of the dialog has been changed from 8pt Tahoma to 10pt Verdana. All scrollbars and other elements of the user interface controls are colored in the same skyblue theme.



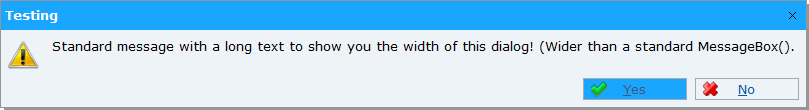
Please note also the redesign of the scrollbars and the comboboxes!.

**Message boxes**

Ever since the MS-Windows Vista update, the standard MessageBoxes of Windows have a new size limit, restricting the width to margin that the development team in Redmont deemed right. If you wished to have more text on a line, you where either on your own, or you needed to dive into the CTaskDialog API.

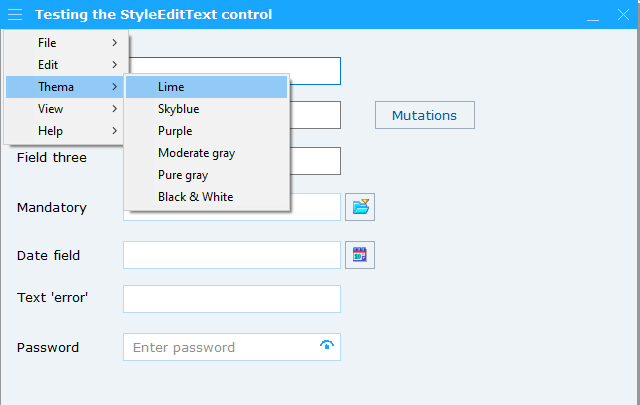
The style framework comes with a StyleMessageBox that’s fixing all this. On top of the fact that all hidden features of the standard MessageBox is there, you can have a few extra’s like:

* Selecting, copying and pasting the text directly from the message box
* Use the ‘do-not-show-again’ registration
* Have the buttons in your own language (currently English, German, French and Dutch)

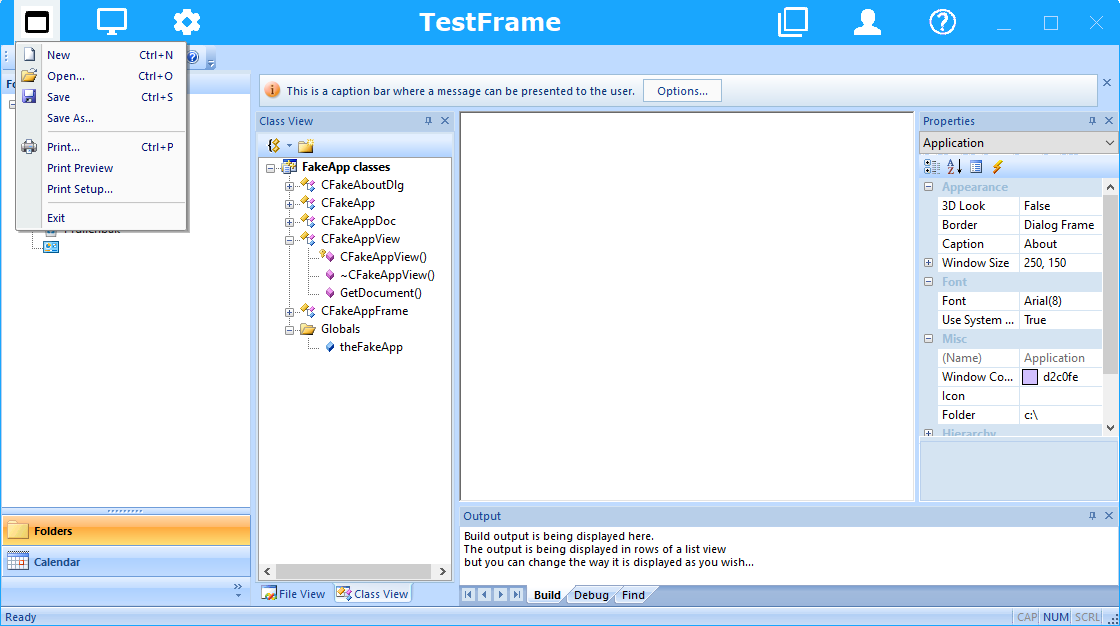


**Menu redesign**

Menu bars (and button bars) take up space in the height. Nowadays displays seems to have pretty much enough pixels in width, but the number of pixels in height is not growing equally fast. By redesigning the menu to a vertical working base menu, we gain a line of working area in the most restrained direction.

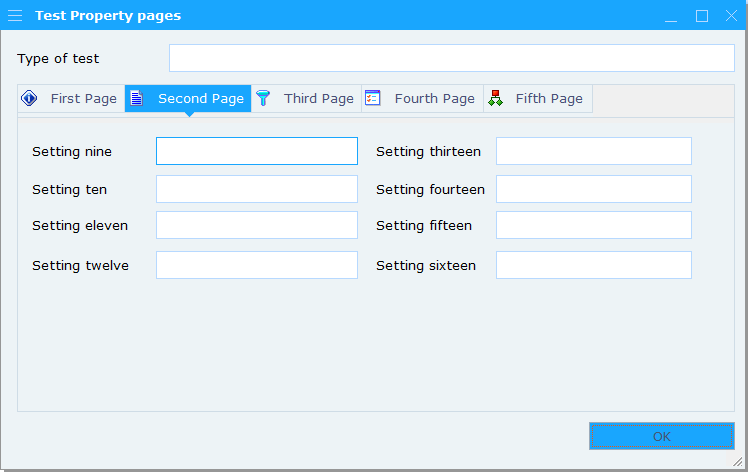


Note that this redesign is not only for dialog based applications, but for the standard MainFrame applications as well. As is shown in a standard MFC wizard setup MDI application in the next image, the menu is now redesign within the non-client title frame, with a couple of menu icons (six in total) these take on the function of the standard menu line.



**Tab controls**

Tab controls are redesigned to function directly with a dialog frame within the pages. Just use the “ InsertItem” method to insert StyleDialog pages into a tab control and you have your property page dialog. No extra coding for CPropertyPages needed!



**Still to write chapters on dialogs**

StyleDialog

StyleMessageBox

StyleFrameWnd

StyleFrameWndEx

StyleMDIFrameWnd

StyleToast

**Still to write chapters on controls**

StyleButton

StyleEdit

StyleCheckbox and StyleRadiobutton

StyleComboBox

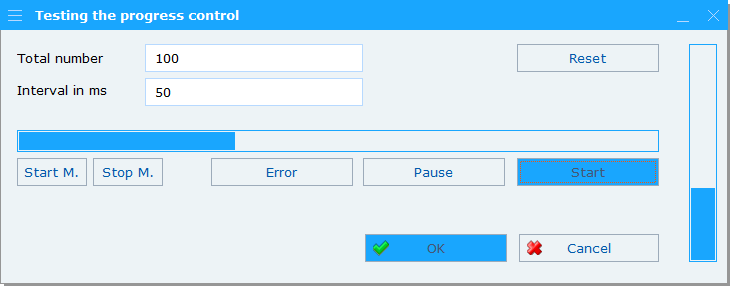
StyleCalendar

StyleHyperlink

StyleListBox / StyleListCtrl

StyleTreeCtrl

StyleProgressCtrl



StyleSpinButtonCtrl

StyleTabCtrl

StyleGridCtrl

**IMPLEMENTING THE FRAMEWORK**

To incorporate the StyleFramework in your MFC project you need to take care of the following steps:

1. Put a copy of the StyleFramework Library project in your own project. Copying the “StyleLibrary” sub-directory in your project and adding the project file to your solution with ‘Add existing project’ should suffice;
2. Make sure that all of your resources in your project are language neutral or in the “English\_UnitedStates” language;
3. Make a copy of the “resource.h” file in your general resource.h files, while taking care of the duplicates. Make sure that your resource-id’s are above 200;
4. Make a link to the StyleLibrary.rc in your projects “<name>.rc2” file, while adding the source path in the file properties of your “<name>.rc” file. This ensures that the resources are linked in;
5. Put an include to “StyleFrameWork.h” in your projects ‘stdafx.h’ or ‘pch.h’ file.
6. Add the include paths to the C++ part of your project file. Make sure you include for all configurations the following under “Additional Include Directories”

$(SolutionDir)StyleLibrary\;$(SolutionDir)StyleLibrary\Grid\

**IMPLEMENTING (NEW) DIALOGS**

Just replacing CDialog with the StyleDialog results in awful effects. To get the proper style as displayed in this document, there are a number of simple steps you must take in existing dialogs, or when implementing a whole new dialog in the dialog editor. These are the steps you must go through in the properties pane of the dialog editor:

1. Change the “Border” attribute from “Dialog Frame” to “None”. Watch the border being removed in the dialog editor, leaving you with a non-bordered dialog;
2. Change the font from “Tahoma;8pt” to “Verdana;10pt”. As soon as you make this change, your dialog will grow substantially with 20% of size;
3. Move all the controls upward till you have a 5 to 7 pt border at the top and 20pt border free at the bottom. The free space will be taken by the framework, as soon as the StyleDialog is initializing;
4. Make sure that all of the controls in the dialog are “Moving type” = “None” and “Sizing type” = “None” !!!

And that’s about it.

Here is an example of the same dialog as displayed on page 2:

