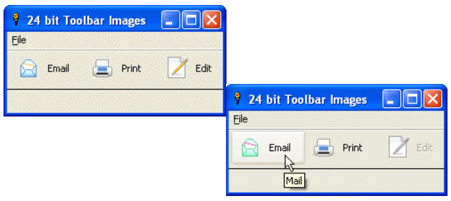
So, you've created your great new program but you think the interface is a little bland. Do you want to spruce it up with one of those nice toolbars with high color buttons that change images when the mouse travels over them? You want the toolbar images to look good even when the button is disabled? You don't want to spend hundreds of hours trying to track down the information needed to accomplish this? You think it should be easy to make a toolbar like that?

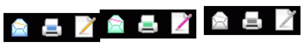
If any of this applies to you then read on. It *is* easy to make a toolbar like that. The hardest part is making the button images. You don't have to use an expensive image editor either; you can make really nice images with free software. MS Paint and The Gimp 2 make a dandy set of image editors that won't cost a thing. You're not an artist you say? No problem, there are free images that you can download and you can use these as they are or modify them to suit. This article won't go into great detail on image creation but it will give a few tips to make it easier for you.

This is one of the toolbars we'll make.



Notice the mouseover color change and the nice disabled Edit button on the right? To create these effects you need to have 3 different images for the toolbar. Really you can create just one, usually the 'cold' image, which is the one that displays for a normal enabled button, and then just apply some coloring, brightness and contrast, or grayscale to make the other two images. These other two images are commonly referred to as the hot and disabled images. The hot image is the one displayed during a mouseover. There are all kinds of effects that you can create for the hot image too, offsets, shadows, whatever you can create with your image software.

These are the cold, hot, and disabled images used in this toolbar.



The cold image was created with Paint, saved as a 24 bit bitmap, then reopened in Gimp to apply the better colors, smudges, and other effects. The size of your image determines the base size of your toolbar buttons. The image above is sized for three 36 pixel x 32 pixel buttons so the final size of each bitmap is 108 x 32. *To make your toolbar bitmaps display correctly the background color must be RGB(1,0,0).* Never use RGB(192,192,192) because it won't be transparent on some systems.

These are the steps that were used to create these three images.

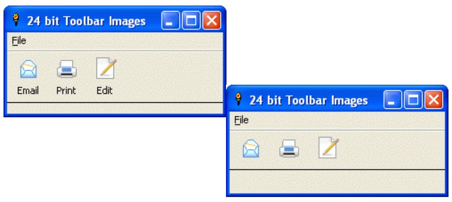
1. Create the cold bitmap with Paint and save as a 24 bit bitmap.
2. Open it in Gimp and apply more colors and effects.
3. Double click the color box: Set the values for R to 1, G to 0, B to 0.
4. Click the BucketFill icon and fill the background.

5. Save it.
6. Create the hot bitmap by adjusting color, brightness and contrast.
7. Fill the background again with RGB(1,0,0).
8. Save it under a new name.
9. For the disabled bitmap, open the cold bitmap in Gimp.
10. Save it under a new name.
11. On the menu: Image >Mode >Grayscale
12. On the menu: Image >Mode >RGB
13. Fill the background again with RGB(1,0,0).
14. Optional: On menu: Filters >Blur >Gaussian blur, set blur radius to 1.0
15. Save.

Tips

- Adding button labels increases the button size.
- Individual button images can be created and then combined into the final toolbar bitmap.
- You can learn a lot about creating these bitmaps by running a program that has some nice toolbar images, like Gimp itself. Press Alt+Print Screen to save the application image, then paste it into a new image. Zoom in and look at how the colors and shapes are used. The Gaussian blur in Gimp makes those soft edged images easily.

Ok, you've created the images, now it's time to create the toolbar and this is where it gets a *little* complicated. The complications are mostly due to the fact that a toolbar is a flexible contraption and there are a lot of options that you can use. Take a look at these.



The only difference between these toolbars and the first toolbar is the *style* used to create the toolbar and whether or not button labels are used.

Toolbar one:
style = _WS_VISIBLE Or TBSTYLE.FLAT Or _WS_BORDER OR TBSTYLE.LIST
Has button labels.
Toolbar two:
style = _WS_VISIBLE Or TBSTYLE.FLAT Or _WS_BORDER
Has button labels.
Toolbar three:
style = _WS_VISIBLE Or TBSTYLE.FLAT Or _WS_BORDER
No button labels.

The toolbar style does not add or remove button labels; it can only position the labels if they exist.

A toolbar isn't much use if you can't tell when your user clicks one of the buttons. A toolbar will send a message to your program when this happens but since LB will ignore these messages you have to trick LB into recognizing them. You do this by creating hidden LB buttons and then assign their IDs to the toolbar buttons. The provided functions take care of the ID assignment so all you need do is create one button for each button in your toolbar, like this.

```
'Buttons for toolbar click events.
Button #form1.tbBtn0, "", Handle.Toolbar.Click, UL, 0, 0, 0, 0
Button #form1.tbBtn1, "", Handle.Toolbar.Click, UL, 0, 0, 0, 0
Button #form1.tbBtn2, "", Handle.Toolbar.Click, UL, 0, 0, 0, 0
```

#form1.tbBtn0 will handle the click for the leftmost toolbar button and the other LB buttons handle the clicks for the toolbar buttons progressing to the right. Put the code to execute for each toolbar button click in the sub or branch of the corresponding LB button.

I've attempted to make toolbar creation easier for you by providing a set of fairly flexible functions. You don't *have* to understand the code involved. You only need to understand the functions and what they are used for. The first thing to do is put your bitmaps into image lists that the toolbar knows how to use. The function for this is named **CreateImageGroup** because it creates from one to three image lists, one for each bitmap, and returns their handles. The lists are used for the normal, hot, and disabled button images. The normal bitmap is required when using this function; the other two are optional.

```
Function CreateImageGroup(byref hImgNormal, bmpNormal$, byref hImgHot,
 bmpHot$, byref hImgDis, bmpDis$, width, height)
```

The function takes 3 strings containing the [path\]filename of each bitmap, bmpNormal$ and optionally bmpHot$ and bmpDis$. The handles of the lists are returned in the byref parameters hImgNormal hImgHot and hImgDis respectively. The remaining parameters are the width and height of the button images. The function is invoked like this:

```
r = CreateImageGroup(hImgNormal, "cold.bmp", hImgHot, "hot.bmp", _
        hImgDis, "disabled.bmp", 36, 32)
```

After the image lists are created invoke the function **CreateToolbar**. This function takes no less than seven parameters.

```
Function CreateToolbar(
hPar,style,btnCount,sep$,hFirstBtn,BtnWidth,BtnHeight,hILn,hILh,hILd)
    'hPar: handle of the parent window that will contain the toolbar.
    'style: toolbar window style.
    'btnCount: number of toolbar buttons to create.
```

```
'sep$: space delimited string of toolbar separator locations, if any.
    'hFirstBtn: handle of the first hidden LB button.
    'BtnWidth, BtnHeight: width & height of the toolbar buttons.
    'hILn: handle to the image list with the normal button images.

'hILh: handle (can be 0) to the image list with the hot button images.

'hILd: handle (can be 0) to the image list with the disabled button im
ages.
```

The function call looks like this when using the image list handles obtained previously.

```
'Create the toolbar.
SeparatorPositions$ = "" 'no separators
TBSTYLE.FLAT = 2048
TBSTYLE.LIST = 4096
style = _WS_VISIBLE Or TBSTYLE.FLAT Or _WS_BORDER OR TBSTYLE.LIST
hToolBar = CreateToolbar(Hwnd(#form1), style, 3
, SeparatorPositions$, _
               Hwnd(#form1.tbBtn0), 36, 32
, hImgNormal, hImgHot, hImgDis)
```

The CreateToolbar function will create the toolbar, assign the image lists, assign the button IDs, enable all of the buttons, and return the handle of the toolbar. A lot has been accomplished so far with just two functions! If you are wondering what the sep$ parameter for CreateToolbar does, change the value of SeparatorPositions$ to "2" and run the sample code.

If you want button labels or tooltips there are methods for these too. Labels can be added by defining a NULL separated string of labels that is terminated with two NULLS. Pass this string to the sub **AddTBlabels** along with the handle of the toolbar.

```
lbl$ = "Email" + chr$(0) + "Print" + chr$(0) + "Edit" + chr$(0) +
chr$(0)
Call AddTBlabels hToolBar, lbl$
```

Tooltips are just as easy. Define a space separated string of text for the tooltips and pass it to the function **AddTooltips** along with the handle of your application window and the toolbar handle. The function returns the handle of the tooltips.

```
tooltip.text$ = "Mail Print Edit"
hTips = AddTooltips(Hwnd(#form1), hToolBar, tooltip.text$)
```

If you need to have a multiple word tooltip for a button just combine the words with an underscore. "Send_Email Print_Document Edit".
The AddTooltips function will convert the underscores to spaces in the tooltips.

Congratulations! The toolbar has been created. Your responsibility does not end here though. The toolbar needs to be told when your window gets resized so it can resize itself to match. Just add this code to your resize handler and it will take care of this task.

```
TB.AUTOSIZE = 1057
Calldll #user32, "SendMessageA", hToolBar As Ulong, _
TB.AUTOSIZE As Long, 0 As Long, 0 As Long, r As Long
```

If you want to disable or enable any of the toolbar buttons you need to call an API function that will take the ID of the button to disable or enable as an argument. An easy way to store the button IDs is to use an array and store the IDs in it. You could use code similar to this. Start with the first hidden LB button that you made for the toolbar.

```
dim btnID(3)
btn1.ID = Hwnd(#form1.tbBtn0)
Calldll #user32, "GetWindowLongA", btn1.ID As Ulong
, _GWL_ID As Long, IDX As Long
btnID(1) = IDX
For i = 2 To 3
    btnID(i) = btnID(i-1) + 1
next i
```

To disable or enable a button get the button ID from the array and use this API function.
Set the enable parameter to True to enable the button or False to disable a button.

```
'Disable button 3
ID = btnID(3)
TB.ENABLEBUTTON = 1025
enable = False
Calldll #user32, "SendMessageA", hToolBar As Ulong, _
1025 As Long, ID As Long, enable As Long, r As Long
```

The last thing your are responsible for is destroying the image list(s) when your application ends. Another sub is provided for this, **ImageListDestroy**. Just call it once for each image list and pass one of the list handles each time, then close your window and end your program.

```
Call ImageListDestroy hImgNormal
Call ImageListDestroy hImgHot
Call ImageListDestroy hImgDis
Close #form1 : End
```

There is no need to destroy the toolbar or tooltips. These are destroyed for you by the operating system.

That's about all there is to it. Only five functions and subs along with a couple of API functions to create, manage, and clean up a nice toolbar with high color 24 bit images.

Sample code

toolbar_with_24-bit_images.zip

- Details
- Download
- 8 KB

-
DennisMcK Jun 19, 2006