Ritchie's QB64 Menu Library

Version 1.0 August 5th, 2012

Thank you for your interest in my menu library. This document was designed to be used as a tutorial and reference. If you take the time to go through the tutorials you'll find that using it will be much easier to comprehend. The tutorial should only take you an hour or two to go through. All example code in this tutorial has been provided as well, but I do suggest typing in the first example as the tutorial progresses through it. You will learn much more if you do this than simply loading the first example in finished format.

The menu library is still very much a work in progress but is still very useful in its present form. I encourage all users of the menu library to show off their work on QB64's forums and highlight any changes, upgrades, modifications and corrections made to the library. If you do make changes to the library please send me a copy of the changes with a brief explanation of what was done. Your changes will be noted in the next release of the library along with credit given to you of course.

You'll find that I've commented almost every line of the menu library for easy understanding and reading. Please don't hesitate to dig around in the code and learn from it as well. I'm not the best QB64 programmer by a long shot, but I believe my code will help programmers that are new to QB64.

I would like to take this opportunity to thank Rob (Galleon), the creator of QB64, for his outstanding contribution to the programming world. A special thanks to all the forum members of QB64 that are so helpful, especially Clippy for maintaining such a great QB64 documentation Wiki. And finally my wife and kids, for listening to the never ending clickety clack of my keyboard over the past few months (even more so now that I have a new Model "M" keyboard!).

If you have any questions, comments, suggestions, flames or concerns please don't hesitate to send them my way, either in the QB64 forum or via email at: terry.ritchie@gmail.com

I hope you have as much fun using the library as I did creating it. Have fun and don't forget to share the programs you create with the library with the QB64 community. ☺

Sincerely,

Terry Ritchie

Note: This document was written as though the reader has general knowledge and use of QB64. If any of the general QB64 commands are unfamiliar or are proving a task to comprehend, turn to the excellent Wiki provided at the QB64 homepage:

~~**http://qb64.net/wiki/index.php?title=Main_Page**~~

https://qb64phoenix.com/qb64wiki/index.php/Main_Page

The Menu Library

A few months ago I was in the process of writing yet another QB64 program when I realized I needed a menu for it. I started out the usual way by sketching out a design, assigning variables to various parts and creating a simple flowchart of how I thought the code should look. After about 300 lines of code I realized designing a menu was no simple task! They look simple enough, sitting there at the top of your screen waiting for input, but I quickly learned that what goes on behind the scenes is anything but simple. It was at this point that I made the decision to create a menu system that was portable with the ability to be added to any QB64 program; the QB64 Menu Library was born.

My goal for the menu library included the following:

- Emulate Windows menus as much as possible.
- Simple to use; no more than two commands to get the menu up and running.
- Make sure the menu is non-destructive to the screen it resides on.
- Full customization of colors, fonts and 2D/3D effects.
- Make it fast and efficient.

I'm happy to report that almost all of my original goals have been met.

- The default menu theme built into the library emulates about 98% of a standard Windows menu. Throughout the documentation I will point out where things deviate a bit.
- Only three commands are needed to get the menu up and running: MAKEMENU, SHOWMENU and CHECKMENU. (I just couldn't get it down to two)
- The menu operates on top of any current screen, restoring the contents underneath.
- The programmer can design a custom menu theme through a myriad of commands that control color, size, font, text location, menu and sub-menu location and 2D/3D look and feel.
- The menu is created with a series of graphics images that can be placed on and removed from the screen very quickly, negating the need to redraw the menu every time it is accessed.

The Menu Library Files

The menu library consists of two main files:

- MENUTOP.BI – a BASIC include file that needs to be located at the very top of your code.
- MENU.BI – a BASIC include file that needs to be located at the very bottom of your code.

We will investigate how to use these files in your code a bit later. For now make sure that all the files from the library ZIP file you downloaded are in your QB64 folder.

When using the library from the QB64-PE Libraries Pack, then you never get in touch with these files.

In the following documentation you can ignore all occurrences of these file names.

In the code examples following below just write $USELIBRARY:'TerryRitchie/MenuLib' in place of the $INCLUDE:'menutop.bi' line and simply omit the $INCLUDE:'menu.bi' line at bottom, that's it.

Constructing a Program

In the QB64 editor type in the following:

```
'$INCLUDE:'menutop.bi'

<a few blank lines>

'$INCLUDE:'menu.bi'
```

MENUTOP.BI contains the type declarations needed for the library, and must always be at the top of your program's code. The file MENU.BI contains the actual functions and procedures that make up the library's command set and must always be the last line in your program's code. If you're wondering what the extension of .BI stands for it's "BASIC Include file".

If you chose to place the library files in a folder other than QB64 then you'll need to modify the above to lines to accommodate this. For example, I keep library files in a folder called "Libs" inside my QB64 folder. I would need to append this folder name as follows:

```
'$INCLUDE:'Libs\menutop.bi'
'$INCLUDE:'Libs\menu.bi'
```

Let's start out by writing a simple menu in as few lines as possible and then investigate the commands used. Modify your code to the following (included as example1.bas):

```
~~'$INCLUDE:'menutop.bi'~~ $USELIBRARY:'TerryRitchie/MenuLib'

~~CONST FALSE = 0, TRUE = NOT FALSE~~ Use compiler constants _FALSE and _TRUE instead.
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
MAKEMENU myMenu$
SHOWMENU
DO
    Menu% = CHECKMENU%(~~TRUE~~) _TRUE instead.
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

~~'$INCLUDE:'menu.bi'~~
```

When you execute the code you should see a simple menu as seen in figure 1 below.

```
myMenu$ = "&File,&Open...#CTRL+O,&Save#CTRL+S,-E&xit#CTRL+Q,*" +_
          "&Help,&About...,*" +_
          "!"
```

Of course you can write it in one line too or split it further into single entries if you want.

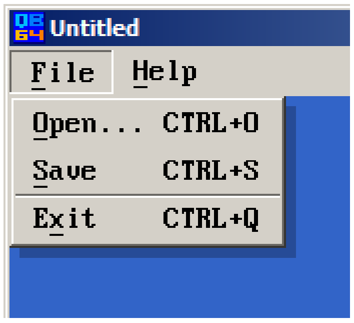

Figure 1 – your first menu

Go ahead and navigate around the menu. It behaves exactly like you would expect any other Windows menu to behave through the use of the mouse, keyboard, ALT key combinations and CTRL key hotkey combinations. The menu theme you are seeing is the default menu theme chosen by the library when no customization has been performed by the programmer. The font used is the standard system font incorporated into all QB64 programs. Let's investigate the commands starting with MAKEMENU.

MAKEMENU() – Create a standard or custom themed menu.

The library command `MAKEMENU` is used to create the final menu graphics that will be used to display the menu onscreen. MAKEMENU is typically the last command called before showing the menu. All customization commands (discussed later) will have been completed before MAKEMENU is called. If no customization has been done, as is the case with our example program, MAKEMENU will take this into account and create the standard theme seen in figure 1 automatically. `MAKEMENU` has the following syntax:

```
MAKEMENU  MenuData$
```

MAKEMENU has no parameters that need to be passed to it, but does at the very minimum need DATA statements that define the menu itself. In our example program three lines of DATA defined the final outcome of the menu. Let's break down the first DATA statement:

```
DATA “&File”,”&Open...#CTRL+O”,”&Save#CTRL+S”,”-E&xit#CTRL+Q”,”*”
```

This line of code defines the "File" menu and its dropdown entries. The first string in quotes, "&File", is always the main menu bar entry. An ampersand (&) modifier character denotes that the next character is the ALT key letter that will activate the submenu under this heading and that letter will have an underscore placed beneath it. Therefore, this menu's sub-menu can be activated by holding down ALT then pressing F. The ampersand modifier is the only modifier allowed to be used on a main menu entry and is optional as well.

Strings in the DATA statement following the first one are considered sub-menu entries. The second string in quotes, "&Open...#CTRL+O", contains the ampersand modifier as well. Using an ampersand modifier in a sub-menu entry allows the user to simply press the letter following the modifier to activate that sub-menu entry. Therefore, it can be used in two ways: by holding down ALT then pressing F then

O, or holding down ALT then pressing F, releasing the ALT and F keys then pressing O. This is the same action you would expect from any standard Windows menu system.

The second string also contains the pound sign (#) modifier. Any text following the pound sign modifier will be right justified on the sub-menu and is considered a hotkey. In this case, holding down the CTRL key and pressing O will activate this sub-menu entry, *but only while the menu is not currently in use*. A listing of valid hotkey combinations is listed below after the chart.

The fourth string in our DATA statement, “-E&xit#CTRL+Q”, contains the dash, “-“, modifier. The dash modifier tells the menu system to draw a line above this sub-menu entry. The dash modifier must be the very first character in the string. As you can see in figure 1, there was a line inserted above this sub-menu entry.

The fifth string in our DATA statement is a single asterisk, “*”, in quotes. This modifier denotes the end of a sub-menu listing. Any string following is now considered the next main menu bar entry. In the example program’s case the next string is “&Help” and will be considered the next main menu bar entry.

This process keeps repeating until a single exclamation point in quotes, “!”, is reached. The exclamation point modifier tells MAKEMENU that this is the end of all menu entries.

The following chart describes the modifiers that can be used in building a menu system and where they are allowed to be used as regular text characters as well.

Modifier	**Modifies**	**Use as text?**	**Description and Use**
&	Both	Neither	Sets the ALT key character and places an underscore beneath the ALT key character.
#	Sub Entry	Main Entry	Right justifies any text that follows and considers the text a hotkey combination. See below for valid hotkey combinations.
-	Sub Entry	Main Entry	Draws a line above the sub-menu entry. The dash delimiter must be the first character in the string data.
*	Neither	Both	Denotes the end of sub-menu entries and must be contained in a string by itself.
!	Neither	Both	Denotes the end of all menu entries and must be contained in a string by itself.

Chart #1 – menu modifiers and their use

Hotkey combinations can only contain a space, plus sign (+) or no space to denote the separation between the hotkey activator and hotkey character. For example, the following hotkey designations are valid: CTRLX, CTRL X, CTRL+X (uppercase, lowercase or a combination of both is permitted). The only hotkey activators allowed are SHIFT and CTRL and there is no distinction between right and left control and shift keys. The hotkey character can be any alphanumeric character, even modifiers allowed as text in the sub-menu entry, and the following:

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, F12, DEL, DELETE, INS, INSERT, HOME, END, PGUP, PAGE UP, PGDN, PAGE DOWN, UP, UPARROW, UP ARROW, DOWN, DOWNARROW, DOWN ARROW, RIGHT, RIGHTARROW, RIGHT ARROW, LEFT LEFTARROW, LEFT ARROW, ~~SCROLL, SCROLLLOCK, SCROLL LOCK, CAPS, CLOCK, CAPSLOCK, CAPS LOCK, NLOCK, NUMLOCK, NUM LOCK~~ (all case insensitive).

The "lock" keys are no longer supported, those are simply unusual for menu hotkeys, as they would switch keyboard behavior each time.

SHOWMENU() – Bring your menu to life.

Once you've defined your menu through DATA statements and used MAKEMENU to create the menu you'll want to display it onscreen at some point. SHOWMENU places the main menu bar at the top of the screen but before it does this it saves a copy of the background that it will cover. The syntax for the SHOWMENU command is:

```
SHOWMENU
```

Once again, this command has no optional parameters to pass and will simply display the menu when executed. The menu bar is not persistent, meaning that if you were to clear the screen using CLS for instance, the top menu bar would not reappear. It's up to the programmer to keep track of when and where to use SHOWMENU to keep the top menu bar persistent when needed. Keep in mind every time you call SHOWMENU the contents of the background behind the bar will overwrite any previous contents that were saved as well. You may need to issue the HIDEMENU command first if you need the contents restored to the screen. *Note that the SHOWMENU command can only be called after the MAKEMENU command has been issued, otherwise you will get an error.*

HIDEMENU() – Put the menu away for safe keeping.

For most programs you'll probably always want the menu to be displayed. But, if you need to reclaim the space used by the main menu bar you can issue the command HIDEMENU to restore the background image that was previously saved with SHOWMENU and remove the menu from screen. The syntax for HIDEMENU is:

```
HIDEMENU
```

HIDEMENU can be issued no matter what current state the menu is in. If the user happens to have a sub-menu open and your program needs to hide the menu, the contents behind the sub-menu and main menu bar will be restored onscreen. Using SHOWMENU and HIDEMENU together can yield some interesting results, such as a menu that appears only when the mouse is at the top of the screen (included as example2.bas) (lines that have been added or changed from now will be **bold**):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
_MOUSESHOW
MAKEMENU
SHOWMENU
DO
    WHILE _MOUSEINPUT: WEND
    IF (_MOUSEY < 2) AND (GETMENUSHOWING% = FALSE) THEN SHOWMENU
    IF (_MOUSEY > GETMENUHEIGHT%) AND (GETMENUACTIVE% = FALSE) AND_
       (GETMENUSHOWING% = TRUE) THEN HIDEMENU
    Menu% = CHECKMENU%(TRUE)
```

```
LOOP UNTIL Menu% = 103
HIDEMENU
DATA “&File”,”&Open...#CTRL+O”,”&Save#CTRL+S”,”-E&xit#CTRL+Q”,”*”
DATA “&Help”,”&About...”,”*”
DATA “!”

‘$INCLUDE:‘menu.bi’
```

The way the above code operates reminds me of the old Sierra adventure game menus and how they operated, or how you can hide the taskbar in Windows. *Note that the HIDEMENU command can only be called after the MAKEMENU command has been issued, otherwise you will get an error.*

CHECKMENU%() - What has the user selected?

Use CHECKMENU% to get information back about which menu item the user selected either by left clicking, pressing ENTER or utilizing an entry’s hotkey combination. CHECKMENU% can also be used to optionally turn off keyboard checking, rendering the mouse the only input method available to the user. There may be times when you do not want the menu grabbing keystrokes, such as when the user may be typing into an input field. The syntax for the CHECKMENU% command is:

`selection% = CHECKMENU%(allowkbd%)`

If *`allowkbd%`* is set to TRUE (-1) the menu will scan the user keystrokes for menu interaction. If *`allowkbd%`* is set to FALSE (0) the menu will only check for mouse interaction. A value of zero (0) will be returned by *`selection%`* if no menu entry has been selected, or the sub-menu ID number that was selected if a selection was made. Main menu entries always have an even hundreds value. In our example program the “File” menu is given the value of 100 and the “Help” menu the value of 200. If more main entries exist then this just continues on, 300, 400, 500, etc.. Sub-menu entries take this value and add their place in the listing to it. So once again, in our example program, the sub-menu entry “Open... CTRL+O” equals 101, “Save CTRL+S” equals 102 and finally “Exit CTRL+Q” equals 103. Likewise, the sub-menu entry “About...” under the “Help” main entry would equal 201. Using this system the menu library allows an infinite number of main menu entries (well, not infinite, but more than you’ll ever need) and up to 99 sub-menu entries for each.

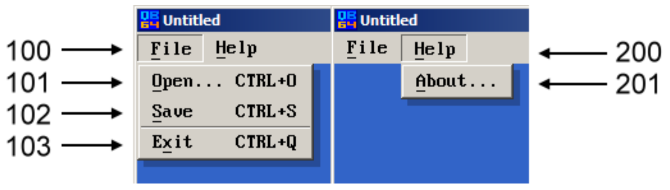

Figure 2 – menu return values

CHECKMENU% will not return the value of the actual main menu entry number but instead only the sub-menu entry number that was selected. There are other commands to see where the user currently is in the menu system which will be discussed later. *Note that the CHECKMENU% command can only be called after the MAKEMENU command has been issued, otherwise you will get an error.*

What's The User Up To?

There are going to be times when you need to know how the user is currently interacting with your menu. The menu library contains three useful commands to determine this.

GETMENUACTIVE%() - Is our user currently accessing the menu?

The GETMENUACTIVE% command will return the status of the user and menu interaction. The syntax for GETMENUACTIVE% is:

```
activity% = GETMENUACTIVE%
```

If the user is currently using the menu system `activity%` will be TRUE (-1), otherwise `activity%` will return FALSE (0). *Note that the GETMENUACTIVE% command can only be called after the MAKEMENU command has been issued, otherwise the value returned will be meaningless.*

GETMENUSHOWING%() - Is the menu currently onscreen?

The second example program showed one possible way to make a menu that only appears when the user wants. The GETMENUSHOWING% command can used to determine if the menu system is currently onscreen. The syntax for GETMENUSHOWING% is:

```
showing% = GETMENUSHOWING%
```

If the menu is currently onscreen `showing%` will be TRUE (-1), otherwise `showing%` will be FALSE (0). *Note that the GETMENUSHOWING% command can only be called after the MAKEMENU command has been issued, otherwise the returned value will be meaningless.*

GETCURRENTMENU%() - Which menu is the user currently on?

GETCURRENTMENU% gives you the ability to track the user's mouse movement or keyboard navigation across the menu system in real-time. The syntax for the GETCURRENTMENU% command is:

```
current% = GETCURRENTMENU%
```

The current menu ID number will be returned in `current%`. This also includes the ID number of a main menu entry contained in the top bar. In our example program, for instance, if the user's mouse pointer is currently on the "File" entry in the top bar, `current%` would contain the value of 100, even if the entry's sub-menu is currently showing. This command could be used to populate an information bar at the bottom of the screen with useful information about the current menu option highlighted. You might wish to create those pesky little yellow information pop-ups at the current mouse location instead. *Note that the GETCURRENTMENU% command can only be called after the MAKEMENU command has been issued, otherwise the returned value will be meaningless.*

SETMENUSTATE() - Disabling and enabling menu entries

The menu library's SETMENUSTATE command can be used to enable or disable a sub-menu entry. For instance, you may want "Paste" in your "Edit" menu to remain disabled until the user has used the "Copy" menu entry to place something in the clipboard. Once this happens "Paste" now has relevance and should be enabled since there is something in the clipboard to paste. The syntax for the SETMENUSTATE command is:

SETMENUSTATE *menuid%*, *state%*

The value for *menuid%* is the ID number of the submenu you wish to enable or disable. To enable a sub-menu entry set *state%* to TRUE (-1) and to disable a sub-menu entry set *state%* to FALSE (0). *Note that the SETMENUSTATE command can only be called after the MAKEMENU command has been issued, otherwise you will get an error.* Let's go ahead and disable the "Save   CTRL+S" sub-menu using the code from the first example (included as example3.bas):

```
 '$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

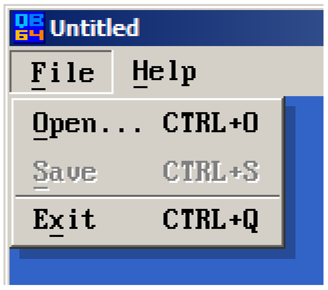

Figure 3 – a disabled sub-menu entry

You'll notice that the default theme still highlights this entry when the mouse pointer is on it, but left clicking yields no results. When using the keyboard the entry is skipped all together.

Custom Menus – A Patriotic Look

The standard Windows default theme is nice, in a boring sort of way, but if you are the creative type you can change all of that. Let's go ahead a build a patriotic theme that Uncle Sam would be proud of. The first thing that needs to get changed is the font.

SETMENUFONT() – Comic Sans anyone?

The SETMENUFONT command allows the programmer to choose an external font file to load and set the size of the desired font. The syntax for SETMENUFONT is:

SETMENUFONT *fontfile$*, *fontsize%* , memory%

Both proportional and monospaced fonts are supported and their location and name are specified in *fontfile$*. If the font you wish to load is located in the current folder, simply supply the name of the font, "lucon.ttf" for example. If the font is located somewhere other than the current directory simply supply the path as well as the name, "fonts\lucon.ttf" for example. The font size specified in *fontsize%* must at least be a value of 1. The menu library will adjust the menu size accordingly.

Windows 7 uses the "segoeui.ttf" font as the default Windows font. This font is included with the library for the educational purpose of this documentation. Let's modify our code from example 3 to load this font using a 16 point font size (included as example4.bas).

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

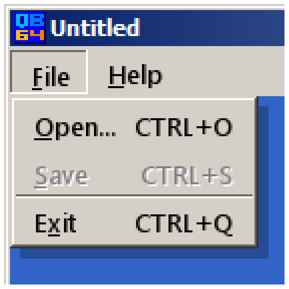

Figure 4 – custom font

While writing this library I learned that not all fonts are created equally. I tried my best to adjust for font location based on size, underscore location and menu heights. After thinking I had the problem solved, another font I would load threw the math off once again. The solution was to gear all the settings towards the default theme built into the library and give tools to the programmer to tweak these settings as needed. When you execute the code above you may notice two things; the menu entry text fields seem too low and the ALT underscore seems a bit too high. This issue becomes more pronounced the larger you make the font. Well, that's not a problem, let's adjust them. *Note that the SETMENUFONT command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

SETMENUTEXT() – Bring the text into line

The SETMENUTEXT command can be used to tweak the position of the menu text vertically. The syntax for the SETMENUTEXT command is:

SETMENUTEXT *tweak%*

Setting a positive value for *tweak%* lowers the text by *tweak%* pixels and setting *tweak%* to a negative value raises the text by *tweak%* pixels. Since our custom font appears to be too low let's tweak the position of the text up a bit shall we (included as example5.bas)?

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"
```

```
'$INCLUDE:'menu.bi'
```

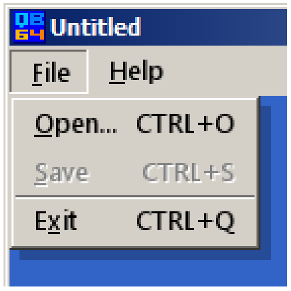

Figure 5 – text tweaked up a bit ... that's better

Now that we have the text in a better position that ALT underscore still looks a bit too high. Let's take care of that as well. *Note that the SETMENUTEXT command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

SETMENUUNDERSCORE() – Move that line

SETMENUUNDERSCORE is used to tweak the ALT underscore vertically. The syntax for the SETMENUUNDERSCORE command is:

SETMENUUNDERSCORE *tweak%*

Just like the SETMENUTEXT command, supplying a positive value for *tweak%* moves the ALT underscore down by *tweak%* pixels while supplying a negative value for *tweak%* move the underscore up by *tweak%* pixels. Since our underscore appears to be too high, let's lower it a bit (included as example6.bas):

```
 '$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

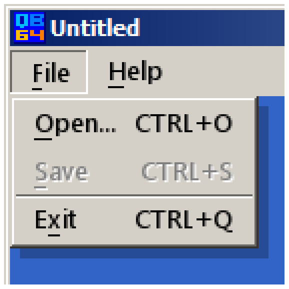

Figure 6 – underscore lowered

Our menu now looks much better thanks to our tweaking tools. *Note that the SETMENUUNDERSCORE command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

Menu Heights

The menu library takes the font size then adds 4 pixels above and below to obtain the menu height. I've found that this formula works well most of the time, but not always. Included with the library for educational purposes is the monospaced font "lucon.ttf" to illustrate this. Let's change our code once again to load this font and see the results first-hand (included as example7.bas):

```
 '$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "lucon.ttf", 16
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

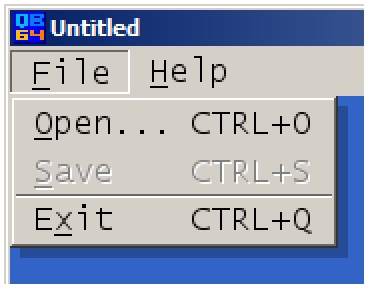

Figure 7 – font seems too big for menu

As figure 7 above illustrates, the font appears to be too large for the given menu height. Well, time to do some tweaking again.

SETMENUHEIGHT() – Reaching new heights

The SETMENUHEIGHT command allows the programmer to force the height of the menu entries. Again, the menu library adds the value of eight (8) to the font size selected. So, if the menu height seems too small you need to adjust it higher than this value. If the menu height seems too large, you can adjust lower than this value. The syntax for the SETMENUHEIGHT command is:

`SETMENUHEIGHT` *`height%`*

The amount provided in *`height%`* will force the menu entries to *`height%`* size in pixels. In the previous demo code we specified a font height of sixteen (16). The menu library added eight (8) to this for a total menu height of 24 pixels. This number however, as illustrated by figure 7 above seems to be too small, making the text appear to be forced in. There may be times when a menu like this is desirable, but let's assume this is not one of those cases. Let's add three (3) more pixels above and below for a total menu height of 30 (included as example8.bas):

```
 '$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "lucon.ttf", 16
SETMENUHEIGHT 30
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

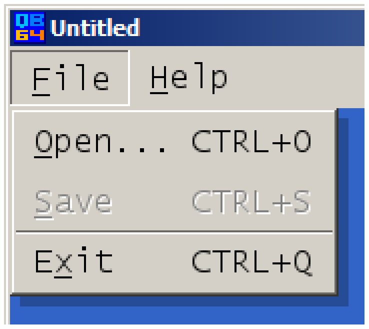

Figure 8 – that's better ... a little breathing room

As figure 8 shows, that little extra height made a big difference. Notice how the lucon.ttf font centered itself properly but the ALT underscore line might be a bit too high? This is a perfect example of what I was stating earlier about how all fonts are not created equally. *Note that the SETMENUHEIGHT command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

ROYGBIV – Splash some color.

Shades of gray might work for Microsoft (and the Navy) but I'm guessing you'll want to spice up your menus with come color. The menu library offers two commands to control menu color, one for the main top bar menu and another for the sub-menus. Let's start by setting the top menu bar text and background colors.

SETMAINMENUCOLORS() – Give that top bar a little pizzazz

The SETMAINMENUCOLORS command is used to customize the text and background colors of the top menu bar. There are three color schemes that need to be set; normal, highlight and selected. The syntax for the SETMAINMENUCOLORS command is:

```
SETMAINMENUCOLORS atext~&, htext~&, stext~&, aback~&, hback~&, sback~&
```

Take note that all the information passed to this command must be of the unsigned long integer type (~&). This is because we are dealing with colors and QB64 requires them to be unsigned long integers. The following inputs are needed and their meaning:

atext~& - the active, or normal, text color. (the menu is sitting idle)
htext~& - the highlighted text color. (mouse is hovering over a top menu entry)
stext~& - the selected text color. (user has left clicked a top menu entry)
aback~& - the active, or normal, background color. (the menu is sitting idle)
hback~& - the highlighted background color. (mouse is hovering over a top menu entry)
sback~& - the selected background color. (user has left clicked a top menu entry)

Let's modify the 6th example a bit to get some patriotic red and white in the top bar menu (included as example9.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMAINMENUCOLORS _RGB32(127, 127, 127), _RGB32(191, 191, 191),_
                  _RGB32(255, 255, 255), _RGB32(127, 0, 0),_
                  _RGB32(191, 0, 0), _RGB32(191, 0, 0)
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

Using SETMAINMENUCOLORS the text has been set to get progressively brighter throughout the menu sequence. Sitting idle the text will be gray (127,127,127). When the user moves the mouse over the text it will brighten up a bit (191,191,191) and finally when the user selects the text it will go full white (255,255,255). Likewise, the menu bar background will change according to user actions. Sitting idle the menu bar is dark red (127,0,0) but when the menu has been activated the background color brightens up to (191,0,0) for both the highlighted and selected conditions.

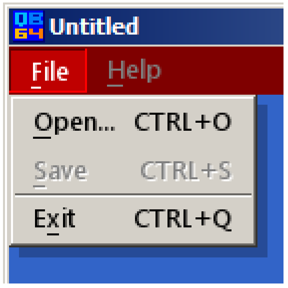

Figure 9 – seeing red

Try to avoid using full brightness for background colors when the menu is in 3D mode as shown in figure 9 above. The menu library calculates a lighter and darker color for the given background to create the 3D effect. If a full brightness color is used the 3D effect would get washed. This is not a problem when the menu is in flat (non-3D) mode (changing 3D modes will be discussed later). *Note that the SETMAINMENUCOLORS command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

In order to be truly patriotic we need to add some blue in there somewhere. The sub-menus are a perfect candidate for this.

SETSUBMENUCOLORS() – Drop down a bit of color

The SETSUBMENUCOLORS command is used to customize the text and background colors of the sub-menus. There are four color schemes that need to be set; normal, highlight, inactive and inactive highlight. The syntax for the SETSUBMENUCOLORS command is:

```
SETSUBMENUCOLORS atext~&, htext~&, itext~&, ihtext~&, aback~&, hback~&,
ibacK~&, ihback~&
```

Take note that all the information passed to this command must be of the unsigned long integer type (*~&*). This is because we are dealing with colors and QB64 requires them to be unsigned long integers. The following inputs are needed and their meaning:

atext~& - the active, or normal, text color. (the sub-menu is sitting idle)
htext~& - the highlighted text color. (mouse is hovering over a sub-menu entry)
itext~& - the inactive text color. (the sub-menu entry is disabled)
ihtext~& - the inactive highlight text color. (sub-menu entry disabled and mouse hovering over it)
aback~& - the active, or normal, background color. (the menu is sitting idle)
hback~& - the highlighted background color. (mouse is hovering over a sub-menu entry)
iback~& - the inactive background color. (the sub-menu entry is disabled)
ihback~& - the inactive highlight background color. (entry disabled and mouse hovering over it)

Let’s make another change to our last example to add the blue component we need for our patriotic menu (included as example10.bas):

```
 ‘$INCLUDE:‘menutop.bi’

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT “segoeui.ttf”, 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMAINMENUCOLORS _RGB32(127, 127, 127), _RGB32(191, 191, 191),_
                  _RGB32(255, 255, 255), _RGB32(127, 0, 0),_
                  _RGB32(191, 0, 0), _RGB32(191, 0, 0)
SETSUBMENUCOLORS _RGB32(191, 191, 191), _RGB32(255, 255, 255),_
                 _RGB32(64, 64, 64), _RGB32(95, 95, 95), _RGB32(0, 0, 127),_
                 _RGB32(0, 0, 191), _RGB32(0, 0, 127), _RGB32(0, 0, 159)
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
```

```
HIDEMENU
DATA “&File”,”&Open...#CTRL+O”,”&Save#CTRL+S”,”-E&xit#CTRL+Q”,”*”
DATA “&Help”,”&About...”,”*”
DATA “!”

‘$INCLUDE:‘menu.bi’
```

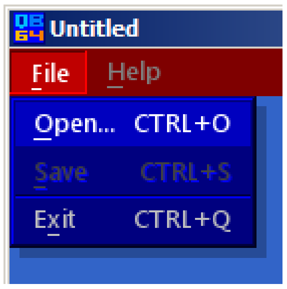

Figure 10 – old glory?

And there we have it; SETSUBMENUCOLORS has changed our once bland submenu to something only a senator could love. *Note that the SETSUBMENUCOLORS command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

I agree that the patriotic menu looks a bit horrid. The colors were chosen to highlight how the different areas of the menu and sub-menu are affected by the SETMAINMENUCOLORS and SETSUBMENUCOLORS commands. Carefully planning out a color scheme for your menu can make all the difference to the look and feel of your program.

What About Flatland?

As you’ve seen, the menu library defaults to a partial 3D scheme by setting the main menu entries in 3D as well as the entire sub-menu box. Disabled sub-menu entries are given the look of being “punched into” the background as well.

SETMENU3D() – How do I look?

The SETMENU3D command is used to control four menu areas to give a 3D or flat look; menu top bar, main menu entries, sub-menu box and sub-menu entries. The syntax for the SETMENU3D command is:

`SETMENU3D` *`topbar%`*, *`submenubox%`*, *`mainentries%`*, *`subentries%`*

`topbar%` - TRUE (-1) for a 3D menu bar, FALSE (0) for a flat look (default = FALSE)
`submenubox%` - TRUE (-1) for a 3D sub-menu box, FALSE (0) for a flat look (default = TRUE)
`mainentries%` - TRUE (-1) for 3D main entries, FALSE (0) for a flat look (default = TRUE)
`subentries%` - TRUE (-1) for 3D sub-menu entries, FALSE (0) for a flat look (default = FALSE)

I’ve never seen a Windows menu that had a 3D top menu bar or 3D sub-menu entries but the options are available to play with in the menu library. Let’s take example 6 again and modify its 3D settings to see the outcomes (included as example11.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMENU3D TRUE, TRUE, TRUE, FALSE
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

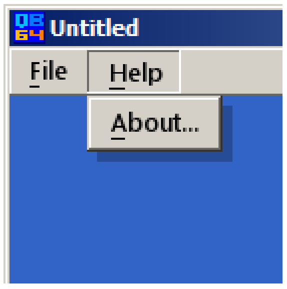

Figure 11 – top menu bar in 3D

By setting the fist parameter of SETMENU3D to TRUE (-1) the top menu bar now has a raised 3D look. Change the SETMENU3D line to:

```
SETMENU3D TRUE, FALSE, TRUE, FALSE
```

then execute the program again.

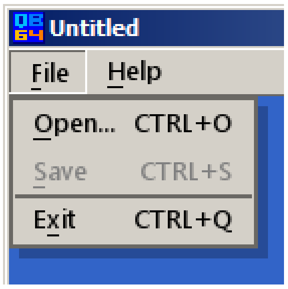

Figure 12 – welcome to flatland

By setting the second parameter of SETMENU3D to FALSE (0) all 3D aspects of the sub-menu box have been removed. The separator line is no longer in 3D, disabled menu entries no longer looked "punched in" and a solid line is now drawn around the sub-menu box itself. Let's turn off the main top bar 3D and turn off the 3D effect main entries and see what we get. Change the SETMENU3D line to:

SETMENU3D **FALSE**, FALSE, **FALSE**, FALSE

then execute the program again.

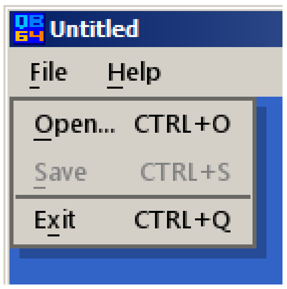

Figure 13 – plain main menu

By setting the third SETMENU3D parameter to FALSE you can see that the main menu entries don't change at all using the default color scheme. The 3D outward button when hovering and inward button when clicked are no longer present. Furthermore, the text does not move slightly when clicked or hovering. The only way you'll be able to give your users any indication that there is menu interaction is to use SETMAINMENUCOLORS and supply varied colors for the three states of text, or backgrounds, or both. It may be desirable at times, however, to have a minimalistic menu such as this.

Finally, let's see what 3D sub-menu entries look like. Change the SETMENU3D line to read:

```
SETMENU3D FALSE, TRUE, TRUE, TRUE
```

and then run the code again.

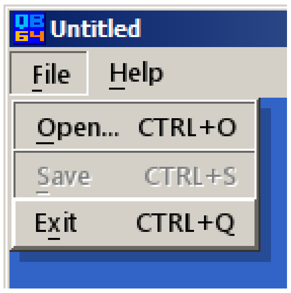

Figure 14 – 3D sub-menu entries

Now the sub-menu entries act like 3D main menu entries. When the mouse is hovered over them they act like a pushbutton. You may have noticed that the default color scheme is changed automatically when 3D sub-menu entries are requested. Instead of the blue background color and white lettering the background and text color remain the same. The only noticeable difference now is, again, the entry acts like a pushbutton and the text shifts slightly to give the impression of a button moving. You can always over-ride this with the use of SETSUBMENUCOLORS to create a custom background color. Also notice that separator lines are of no consequence any longer, since the entire entry is turned into a button.

Play around with the 16 possible combinations that the SETMENU3D command offers to get just the right look and feel. *Note that the SETMENU3D command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

Me and My Shadow

Throughout all of the examples so far you have seen a shadow under the sub-menu boxes. You can adjust the height of the sub-menu boxes for a larger, smaller or no shadow at all.

SETMENUSHADOW() – Dim the lights

The SETMENUSHADOW command can be used to adjust the size of the shadow under sub-menu boxes by supplying a height value. The syntax for the SETMENUSHADOW command is:

SETMENUSHADOW *height%*

The value of height specifies how large the shadow should be. Basically, it's the number of pixels to the right and bottom of the sub-menu boxes the shadow protrudes out. The default value is 5 pixels but let's go ahead and change that value to see the results (included as example12.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMENU3D FALSE, TRUE, TRUE, TRUE
SETMENUSHADOW 10
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

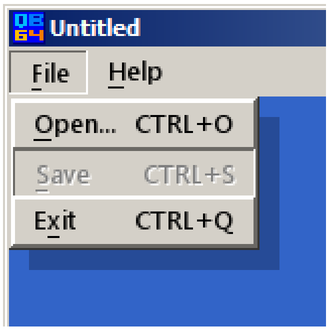

Figure 15 – a 10 pixel shadow

The shadow will also darken any image underneath for a true shadow look. Go ahead and change the value in SETMENUSHADOW to zero (0) and see the results.

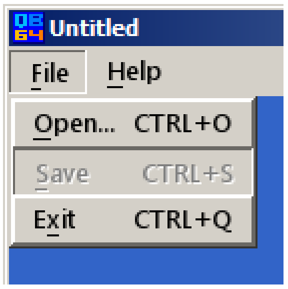

Figure 16 – look ma ... no shadow

Setting SETMENUSHADOW to zero (0) effectively removes the shadowing effect from the menu system. *Note that the SETMENUSHADOW command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

Moving Menus

The menu library includes a few commands to move the placement of the main menu text by indenting X amount of pixels and to raise or lower the sub-menu boxes.

SETMENUINDENT() – Move that text

The SETMENUINDENT command allows you to move the main menu bar text locations to the right. The syntax for the SETMENUINDENT command is:

```
SETMENUINDENT indent%
```

The main menu text entries will be moved by *indent%* pixels to the right. Let's take the previous example and indent the main menu entries by 25 pixels (included as example13.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
```

```
_SCREENMOVE _MIDDLE
SETMENUFONT “segoeui.ttf”, 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMENU3D FALSE, TRUE, TRUE, TRUE
SETMENUSHADOW 10
SETMENUINDENT 25
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA “&File”,”&Open...#CTRL+O”,”&Save#CTRL+S”,”-E&xit#CTRL+Q”,”*”
DATA “&Help”,”&About...”,”*”
DATA “!”

‘$INCLUDE:‘menu.bi’
```

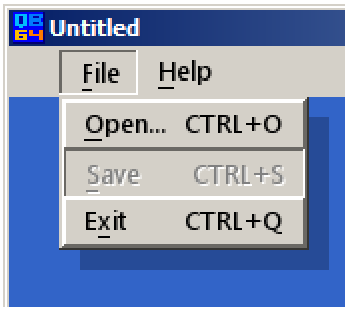

Figure 17 – menu indented 25 pixels

As figure 17 above shows, the SETMENUINDENT command moves the entire menu system to the right. *Note that the SETMENUINDENT command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.*

SETSUBMENULOCATION() – The floating mountains of Pandora

The menu library also has a command that allows the sub-menu boxes to be raised and lowered. The SETSUBMENULOCATION command can be used to accomplish this. The syntax for the SETSEBMENULOCATION command is:

```
SETSUBMENULOCATION tweak%
```

All sub-menu boxes will be moved by *tweak%* pixels. If you supply a positive value for *tweak%* the sub-menu boxes will be lowered by *tweak%*. If you supply a negative value for *tweak%* the sub-menu boxes will be raised by *tweak%*. The default location for sub-menu boxes is just below the main menu bar. I’ve seen some Windows programs have their sub-menu boxes raised a bit, but never seen floating sub-menu boxes which the library will allow for an interesting effect. Left modify the last example once again and create floating sub-menus (included as example14.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMENU3D FALSE, TRUE, TRUE, TRUE
SETMENUSHADOW 10
SETMENUINDENT 25
SETSUBMENULOCATION 25
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

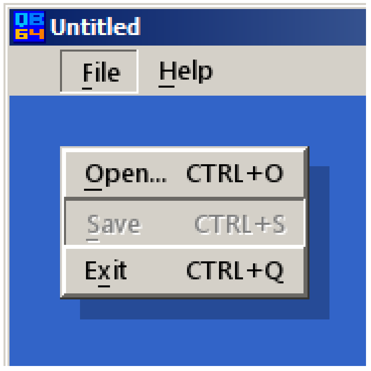

Figure 18 – floating sub-menus

Note that the SETSUBMENULOCATION command can only be called before the MAKEMENU command has issued, otherwise it will just be ignored.

GETMENUHEIGHT%() - How tall is this thing?

There may be times when you need to know the height of the top bar menu in pixels. Again, the menu library takes the font height and adds 8 by default to get the menu height. The GETMENUHEIGHT% command will report back what this value is. The syntax for the GETMENUHEIGHT% command is:

```
height% = GETMENUHEIGHT%
```

The value returned in *height%* will be the height of the main menu bar in pixels. *Note that the GETMENUHEIGHT% command should only be called after the MAKEMENU command has been issued, otherwise the value returned will have no meaning.*

SETMENUSPACING() - The Final Frontier

The final command in the menu library is one that be used to control the spacing before and after text. By default, the menu library pads all main menu and sub-menu text with ten (10) pixels to the right and left. You can use the SETMENUSPACING command to change this default value. The syntax for the SETMENUSPACING command is:

```
SETMENUSPACING spacing%
```

The value given in *spacing%* will be the number of pixels to the right and left of every text field within the menu system. Let's modify our previous program one more time to see this in action (included as example15.bas):

```
'$INCLUDE:'menutop.bi'

CONST FALSE = 0, TRUE = NOT FALSE
DIM Menu%
SCREEN _NEWIMAGE(640, 480, 32)
CLS , _RGB32(50, 100, 200)
_SCREENMOVE _MIDDLE
SETMENUFONT "segoeui.ttf", 16
SETMENUTEXT -1
SETMENUUNDERSCORE 1
SETMENU3D FALSE, TRUE, TRUE, TRUE
SETMENUSHADOW 10
SETMENUSPACING 30
MAKEMENU
SETMENUSTATE 102, FALSE
SHOWMENU
DO
    Menu% = CHECKMENU%(TRUE)
LOOP UNTIL Menu% = 103
HIDEMENU
DATA "&File","&Open...#CTRL+O","&Save#CTRL+S","-E&xit#CTRL+Q","*"
DATA "&Help","&About...","*"
DATA "!"

'$INCLUDE:'menu.bi'
```

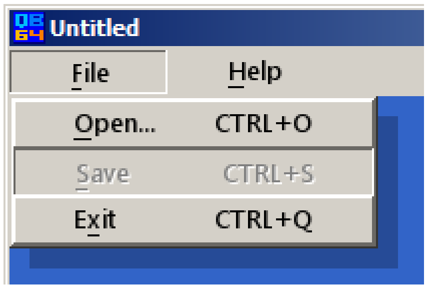

Figure 19 – 30 pixels of spacing

Note that the SETMENUSPACING command can only be called before the MAKEMENU command has been issued, otherwise it will just be ignored.

Well, that's it for version 1.0 of the menu library. The library has been extensively tested for bugs but that's not saying its bug free. If you come across a bug or find that a new command is needed don't hesitate to contact me at terry.ritchie@gmail.com and I'll get working on that bug or new feature. Some future features I would like to implement are:

- The ability to use images as menu backgrounds
- The ability to create cascading menus (sub-menus under sub-menus)
- The ability to save and load various menu customizations (plugin?)
- The ability to change the look of the menu and reissue a MAKEMENU to have the menu change in real-time.

Please feel free to browse through the library's code and change, modify or add anything you wish. If you do implement a new command or feature please email the changes made to me so I can incorporate them into the next version, with credit given to you.

Also, share the programs you create, especially with one of my libraries, on the QB64 website for others to learn from -> ~~www.qb64.net~~ https://qb64phoenix.com/forum/index.php

Terry Ritchie

Command Quick Reference

SUBROUTINES

HIDEMENU

Hide the menu from view.

Only call after MAKEMENU has been called or you will get an error.

MAKEMENU MenuData$

Create the menu system from menu DATA supplied and optional customizations applied.

Call after menu DATA statements have been created and all customizations added.

SETMAINMENUCOLORS *atext~&, htext~&, stext~&, aback~&, hback~&, sback~&*

Set the colors of the main menu bar text and background.

Default:	atext~&	_RGB32(0, 0, 0)
	htext~&	_RGB32(0, 0, 0)
	stext~&	_RGB32(0, 0, 0)
	aback~&	_RGB32(212, 208, 200)
	hback~&	_RGB32(212, 208, 200)
	sback~&	_RGB32(212, 208, 200)

Call before MAKEMENU otherwise settings will be ignored.

SETMENU3D *topbar%, submenubox%, mainentries%, subentries%*

Set the flat or 3D characteristics of the main top bar, sub-menu boxes, main menu entries and sub-menu entries.

Default:	topbar%	0 (FALSE)
	submenubox%	-1 (TRUE)
	mainentries%	-1 (TRUE)
	subentries%	0 (FALSE)

Call before MAKEMENU otherwise settings will be ignored.

SETMENUFONT *font$, fontheight%* , memory%

Specify the font the menu system will utilize and the height of the font.

Default: _FONT 16 (system font)

Call before MAKEMENU otherwise settings will be ignored.

SETMENUHEIGHT *height%*

Overrides the default menu height.

Default: 24 (system font height of 16 + 8)

Call before MAKEMENU otherwise settings will be ignored.

SETMENUINDENT *indent%*

Forces the entire menu system to move to the right.

Default: 0 (zero)

Call before MAKEMENU otherwise settings will be ignored.

SETMENUSHADOW *height%*

Sets the size of sub-menu shadowing effect.

Default: 5

Call before MAKEMENU otherwise settings will be ignored.

SETMENUSPACING *spacing%*

Overrides the default amount of blank space preceding and following menu text.

Default: 10

Call before MAKEMENU otherwise settings will be ignored.

SETMENUSTATE *menuid%, behavior%*

Enables or disables a sub-menu entry.

Default: All sub-menu entries enabled

Call after MAKEMENU or an error will occur.

SETMENUTEXT *tweak%*

Forces the menu and sub-menu entry text vertically by tweak%. Negative values move the text up, positive values move the text down.

Default: 0 (zero)

Call before MAKEMENU otherwise settings will be ignored.

SETMENUUNDERSCORE *tweak%*

Forces the ALT hotkey underscore line vertically by tweak%. Negative values move the line up, positive values move the line down.

Default: 0 (zero)

Call before MAKEMENU otherwise settings will be ignored.

SETSUBMENUCOLORS *atext~&, htext~&, itext~&, ihtext~&, aback~&, hback~&, iback~&, ihback~&*

Set the colors of the sub-menu box text and background.

Default:

Default:	When sub-menu entries in flat mode:	
	atext~&	_RGB32(0, 0, 0)
	htext~&	_RGB32(255, 255, 255)
	itext~&	_RGB32(128, 128, 128)
	ihtext~&	_RGB32(128, 128, 128)
	aback~&	_RGB32(212, 208, 200)
	hback~&	_RGB32(10, 36, 106)
	iback~&	_RGB32(212, 208, 200)
	ihback~&	_RGB32(10, 36, 106)
	When sub-menu entries in 3D mode:	
	atext~&	_RGB32(0, 0, 0)
	htext~&	_RGB32(0, 0, 0)
	itext~&	_RGB32(128, 128, 128)
	ihtext~&	_RGB32(128, 128, 128)
	aback~&	_RGB32(212, 208, 200)
	hback~&	_RGB32(212, 208, 200)
	iback~&	_RGB32(212, 208, 200)
	ihback~&	_RGB32(212, 208, 200)

Call before MAKEMENU otherwise settings will be ignored.

SETSUBMENULOCATION *tweak%*

Forces the sub-menu boxes vertically by tweak%. Negative values move the box up, positive values move the box down.

Default:	0 (zero)

Call before MAKEMENU otherwise settings will be ignored.

SHOWMENU

Places the menu system onscreen.

Call after MAKEMENU otherwise you will get an error.

FUNCTIONS

selection% = CHECKMENU%(allowkbd%)

Retrieves the menu item selected, either by left clicking or pressing ENTER. A return of zero (0) means that no selection was made by the user. Set allowkbd% to 0 (FALSE) to disable menu keyboard checking and -1 (TRUE) to enable keyboard checking.

Call after MAKEMENU otherwise you will get an error.

menuid% = GETCURRENTMENU%

Retrieves the current menu entry the user is active on.

Call after MAKEMENU otherwise the returned value will be meaningless.

active% = GETMENUACTIVE%

Retrieves the state of the menu system. -1 (TRUE) means the menu is currently being accessed by the user, 0 (FALSE) means the menu is sitting idle.

Call after MAKEMENU otherwise the returned value is meaningless.

height% = GETMENUHEIGHT%

Gets the menu height the library calculated from the font height.

Default:	24 (system font height of 16 + 8)

Call after MAKEMENU otherwise the returned value is meaningless.

showing% = GETMENUSHOWING%

Retrieves the menu showing state. -1 (TRUE) indicates the menu is showing onscreen. 0 (FALSE) indicates the menu is hidden.

Call after MAKEMENU otherwise the returned value is meaningless.

TYPE DECLARATIONS

Make sure not to use the following as type declarations, constants or variables:

TYPE MENU

TYPE MENUSETTINGS

ARRAYS

Make sure not to use the following array names as constant or variable names:

Menu() AS MENU
Mset AS MENUSETTINGS