# *Another* Button Bar
# by Light at the End of the Tunnel Software

## About Us

Howdy!  And welcome to **Light at the End of the Tunnel's** first major production.  **Light at the End of the Tunnel** is devoted(?) to producing examples of Visual Basic code for applications that you may have seen and always wondered "How'd they do that?"

If you're like me, you are frustrated by the relative dearth of ready information for the Visual Basic programmer for doing **anything** that requires knowledge of either API routines or the internal workings of Windows and are hungry for any information you can get.  Microsoft was kind(?) enough to give us access to the API but, evidently, was not willing to provide any kind of reference to aid us, the Visual Basic users, in utilizing it.  From what i've seen online and in books, everyone, no matter what programming language they use, has a tough time tracking down correct information about Windows internals, file formats, etc.

So, in an effort to distribute useful (and needed) information, i've produced a few applications (source included) that we hope will enlighten the unenlightened, aid the depraved, heal the crippled, and put the cat out.  Hopefully, we''ve provided enough documentation about the program that you will be able to understand what's going on and use the information.  However, if after a reasonable about of time you still don't get it, feel free to e-mail me with your questions.  And, if you find it useful or learn anything from it, drop me an e-mail and let me know.  Your response could spur the next production.  And, if you **really** like it, and reside in Seattle, and want to hire a PC programmer who's looking to relocate...

Enjoy.

tim koffley
(miami, fl)
[CIS: 70334,16]
[AOL: Tim Servo]

## About the Program

Another Button Bar is a program that turns a program group into a bar of buttons that resides on your "desktop" and allows you to just click on one of the buttons to launch the application that the icon painted on the button represents. There are probably hundreds of these programs "out there" now, shareware and retail, so i'm reasonably sure you can't make any money distributing a new one (ie-you wouldn't gain much). i just wasn't happy with the available "button bar" programs out there and decided to create my own simple one. The real inspiration came when PC Magazine's John Deurbrouck developed one as their free utility of the month. i thought, "Great! At last a cool freebie and i don't have to buy one of the Shareware ones." It **is** great except for one small detail: it doesn't allow you to configure the bar to be positioned vertically as opposed to horizontally. Evidently, that's no small trick in C (using the author's method anyway) and was more trouble than the free utility was worth. So we decided to roll own own (so to speak).

The required tools for using/modifying the program are as follows:
-Visual Basic 3.0, Pro Edition (there is a way around using the Pro Edition i'll mention later)

If you just want to run the compiled program, you need VBRUN300.DLL, THREED.VBX, and PICCLIP.VBX.

The main hurdle was figuring out the structure of .GRP files in Windows. Everything I eventually learned had to be composited from a host of different sources, all of which had either conflicting or missing information. If you dare to try to figure it out for yourself, there is a list of sources i used later on. After figuring out the .GRP file structure, the rest was pretty easy. It only requires a bit of API knowledge to draw icons and run applications.

## **Installation**

Copy TKBAR.EXE to anywhere you want. All of the .VBX files belong in the Windows system directory, usually C:\WINDOWS\SYSTEM. Then use Program Manager to create a new program item for TKBAR.EXE.

If you want to run the program interactively just use TKBAR.EXE. This brings up a dialog that let's you select a .GRP file to load as a button bar. If you're not sure exactly which group is in a file, just click the "Show Name" button and the full group name will be displayed at the top of the form.

Alternately, you can setup an association using the File Manager. Under the File menu is a choice called Associate. Enter GRP for the file type, then Browse to find TKBAR.EXE. Now you can just double-click on a GRP file in the File Manager and it will launch a button bar for that group. Once you've set up an association, you can have Windows load a group of your most-used icons each time you start Windows. To do so, add a "Run=" entry to the WIN.INI file. For instance, "Run=c:\windows\mostwant.grp" will bring up that group when you start Windows. If you want additional button bars to load, just add them after the first file name and separate them with commas. You could also do the same sort of thing with an item in the StartUp group. Just make a copy of the Icon for TKBAR.EXE and place it in the StartUp group and specify a GRP file as a parameter for the icon properties.

Another Button Bar keeps its INI file in the Windows directory.

## **Configuring**

Once the buttons are on the desktop, you can click with the right mouse button on any of the active buttons to popup a menu. Click "Configure" on the menu to bring up a configuration dialog. Here you can set exactly how you want the grid of buttons to look by specifying the number of rows and the number of columns in the grid. You can also specify if you want the window to always be on top of every other window by checking the "Always On Top" check box.

That's all you need to know to run it. If you're interested in the behind-the-scenes stuff, read on.

## About .GRP files

.GRP files are files that Windows uses to keep information on program groups used by the Program Manager.  You should see some files in your main Windows directory that have .GRP extensions.  Basically, they contain all the information you enter when you create a new program group or item as well as information about how to display the group window.

Fortunately, they also contain the actual icons used in a group in a device-dependent format.  What this means is that we don't have to figure out how to draw the icons for different Windows display resolutions, etc.  If you change the display driver you're using, Windows rebuilds all the program group files to adjust to the new driver; all the work is done for us!

If you are new to using user-defined types in Visual Basic you should definitely learn about them.  Most everything done in Windows relies on data structures that organize data into "packages" of related information.  C calls them structures, VB calls them user-defined types, all good programmers call them the best thing since sliced bread (OOP notwithstanding).  One neat advantage (for VB) to defining types is that you can read a bunch of items at once from a file just by reading in the structure as opposed to reading each and every item one at a time.  GRP file data is organized into structures.  The structures we use are defined in TKBAR.BAS in the declarations section.  I'll explain the items that we used in each structure.

**GroupHeaderType**
OffsetTag is the byte position in the file of "tag" information (more on this later).
OffsetName is the byte position of the group name.
LogPixelsX is the width of an icon in the group in pixels.
LogPixelsY is the height of an icon in the group in pixels.
BitsPerPixel used for drawing the icons.
Planes used for drawing the icons.
NumItems the number of pointers to item data structures in the file.

**GroupItemType**
pt contains the coordinates of a program item in the group window.
ANDPlaneBytes is the number of bytes to retrieve for the icon's AND plane from the file.
XORPlaneBytes is the number of bytes to retrieve for the icon's XOR plane from the file.
OffsetANDPlane is the byte position of the icon's AND plane.
OffsetXORPlane is the byte position of the icon's XOR plane.
OffsetName is the byte position of the program item's name.
OffsetExeName is the byte position of the executable name.

**TagDataType**
Ok.  i only called this stuff "tag" data because that's what J. Deurbrouck called it in his BTNGO program.  This is a structure used to retrieve information from a linked list in the group file.  The linked list contains executable pathnames and program item state information.  i'm not entirely certain about how this structure is used but was able to deduce enough to get it to work.  Read on for a better explanation.

**<u>Reading the file...</u>**
(Note: This was meant to be read as a companion to the source code. See procedures GetItemInfo & ButtonBarInit)
Alright, now that we've got some structures defined, we can start reading in information from the file. The first thing in the file is the GroupHeader structure. One of the items in this header is NumItems which is a count of the number of pointers (to GroupItem structures) that immediately follow this structure. So say GroupHeader.NumItems is 7. This means that we now read in 7 item pointers. We'll call them pointers but they're really byte offsets to each of the GroupItem structures for each program item. Now we've got 7 pointers to 7 GroupItem structures in the file, well...maybe. Because of the way Windows manages the group file, it just zeroes out a pointer if you delete a program item. What this means to us is that we ignore any pointer that is zero and that the number of valid program items in the file is equal to or less than GroupHeader.NumItems. Anyway, now we've got pointers/byte positions for the GroupItem structures.

(Just so I don't have to keep restating it, any structure item that starts with the word "offset" is a byte position in the file. Also, the first byte position in a file is position zero. This is different than Basic which treats the first position as position one.)

Now, for any item pointer that isn't zero, we seek to that byte position and read in a GroupItem structure. As you can see from the explanation of this structure, this structure is mainly a map to the real information we need for each program item. So all we have to do is read the map, go to each location, and retrieve the "guts" of the program item. It's fairly straightforward.

One thing that might not be clear is retrieving strings from the file. The strings are items like the group name, the group item name, and the various paths and filenames associated with each item. Strings in the file are null-terminated which means, simply, that the end of the string is signalled by an ASCII value zero. So to read in a string, you seek to the start position of the string, and read in characters until you hit an ASCII zero. Simple!

There are two items we read as strings but not the way we just mentioned. They are the AND plane and the XOR plane for the icon. For these we are explicitly given the number of bytes for each in the GroupItem structure. All we do is create an empty string buffer that equals that length and read it in using the usual Basic binary read.

**<u>Reading "tag" data</u>**
There's one last <u>crucial</u> piece of information that we need before we can do anything and that is the path where each executable resides. This is what drove me nuts (see annotated references). The fact is, the item called OffsetExeName in GroupItem is a pointer to a string (in the file) that is as follows: the working directory for the program plus the executable name plus any command line arguments. Note that **none** of these is the path where the executable resides. You have to dig for that information!

There's an item in GroupHeader i called OffsetTag that is a pointer to a linked list towards the end of the file. A linked list is simply a data structure where each item contains a pointer to the next item in the list, like a train. To find where the next item is, you just look at the pointer in the current item. This is how the "tag" data is arranged.

You read an item, see if there's additional stuff you should read, read it, go to the next item, then repeat these steps until the end of the list is signalled. The field in TagData called Id is how we know what to do for each item in the list. If it's one value, we know to read additional info, if it's another, we know we've come to the end of the list. The additional stuff we read is an executable path name. The item to which it applies is identified by the current TagData.Item. Group items are numbered sequentially based on their pointer positions (e.g. the item pointed to by the third pointer is item number 3).  You should be able to walk through the code and figure out what's happening.

We now have everything we need to tell Windows how to execute each program item.

## <u>Miscellaneous</u>

You only need PICCLIP.VBX because of my fancy little "About" popup. It's not **really** necessary but i thought the animation looked cool enough to include it. And who knows, someone might learn how to do simple animation because of it. Also, VB does some strange things with storing the Picture property of controls so you will most likely get a **load error** when you load the TKABT.FRM. The solution is to re-set the Picture property of PicClip1 to be TUNNELS2.BMP. And make sure that its Cols property is 4 and its Rows property is 1. A similar problem occurs witht the Icon property of the Button Bar form. i set that to BAR.ICO instead of the default form picture.

As i mentioned earlier, if you don't have the VB Pro Edition controls, you can get around it. One possibility is to change the 3D push buttons into Picture boxes. You shouldn't have to rewrite much code with that change. Most of the rewrite would be for drawing the buttons. Also, you can simulate the 3D control by having dark grey and white lines drawn around each button that become visible/invisible depending on the MouseDown/Up events.

One semi-unconventional thing in Another Button Bar is my choice of using the often overlooked right mouse button to popup the configuration menu. i was forced into this for two reasons: 1) It's not easy (read nearly impossible) to add items to the control menu for a window and respond to them correctly. VB does it's own menu management and doing anything with menus outside of VB is real touchy. 2) Because of #1, the best way to safely do menus in VB is to have a menu line. That would have looked awful in the vertical configuration and just plain would have used more screen space. So that left us with a popup menu and the only trigger left, the right mouse button. i think a lot of programs out there overlook the right button or don't advertise it's use very well in documentation. It's every bit as good as the left button and a fine button in its own right. Come on! Let's hear it for the right mouse button!

## **References**

Visual Basic Programmer's Guide to the Windows API: Daniel Appleman, Ziff-Davis Press
This book is a godsend.  It's basically the book MS should have provided VB programmers but didn't.  If you are doing anything with API in VB you should own this book.  If you aren't doing anything with API in VB, you're missing out on a lot of power.

PC Magazine:  Sept. 14, 1993, Volume 12, Number 15   "Utilities: BTNGO" by John Deurbrouck
As i mentioned, this (or rather it's lack of vertical capability) was the inspiration for Another Button Bar.  If you are a member of ZiffNet you can download the C source code from the PC Mag Utillities library.  He did one thing in the code that threw me off; he made his own structure for keeping program item information and used the name IconPath for one of the structure items. Problem is, he ultimately put the executable paths into IconPath so i give him points off for bad variable naming.  However, he gets points for showing us how to read the executable paths from the group file, without which there would be no Another Button Bar.

MS Developers Network on CIS,  ShowGrp.C example,  GO MSDNLIB, Filename: 4-20.ZIP
This is a C example on how to read the group files.  Their contribution to the puzzle was completely omitting information on reading the executable paths.  More of the code is for formatting output windows than for actually reading the files.  Still, it's another group file example if anyone's interested.

## Disclaimer

No warranty exists, either express or implied.  No liability is assumed for any damage or loss resulting from the use of this program.

This software is provided as Freeware, i.e. it is free of charge.  You owe no money for the use of this program.  As such, i make no warranties or guarantees regarding its use and performance.  i'm reasonably confident that the program will not cause harm to your PC or any files residing on it, but in the event of a breakdown in cosmic probabilites, we are **not** responsible.  In the extremely unlikely event that it does cause harm, please let me know so that i might prevent the program from causing any further damage.  Also, if something occurs that appears to be a major blunder, please let us know.

Hi-Keeba!
Tim Koffley

## Version History

| | |
|---|---|
| 1.0 | 11/93 Initial release |
| 1.0a | 12/93 Fixed medium-sized bug that would freeze machine.  Try running the old version on a group where no program item had a directory prefixed to the EXE name. (Then hard reset your machine <g>). Got rid of APIGUIDE.DLL. |