Ritchie's QB64 Graphics Line Input (GLINPUT) Library

Version 2.10 June 14th, 2012

Thank you for your interest in my GLINPUT 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 the tutorials 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 GLINPUT library is still very much a work in progress but is still very useful in its present form. I encourage all users of the GLINPUT library to show off their work in 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 every line of the GLINPUT 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 SMcNeill for starting a thread on the QB64 forum that resulted in my creating this library and a very special thank you 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.

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

LINE INPUT

The LINE INPUT command has been a staple of Microsoft flavored BASIC for as long as I can remember (and I've been using them since 1979!). LINE INPUT can be used two different ways; as a method of inputting text from the user or as a means of acquiring a string from a file or device. This library was created to mimic an enhanced version of LINE INPUT as a method of gathering text from the user. The GLINPUT commands cannot be used to retrieve string data from files or devices.

LINE INPUT when used as a method of gathering text from the user has a few drawbacks. The GLINPUT library was created to overcome these limitations. The limitations that the library specifically addresses are:

- Programs must sit idle at LINE INPUT until the user presses the ENTER key
- LINE INPUT does support the major editing keys, such as INSERT, ARROW, and DELETE keys
- Only one LINE INPUT can be shown and active on the screen at any given time
- LINE INPUT can't use graphics coordinates as only text coordinates are allowed
- Once text has been entered there's no easy method of removing the command from the screen.

The GLINPUT library overcomes these limitations with the following features:

- Text input can be performed regardless of background operations; your program continues.
- All major editing keys are supported.
- Multiple inputs can be placed on the screen allowing the user to freely move between them.
- Inputs can be placed on graphics screens at pixel coordinates (SCREEN 0 not supported).
- Pressing ENTER is not required to retrieve the contents of an input field.
- Once input is finished the command can easily be removed from the screen.

The following features are also offered by using QB64's enhanced text manipulation commands

- Multiple fonts, font sizes, foreground and background colors are supported all at the same time.
- Multiple destination image handles supported.
- Nondestructive to the background, even animated ever changing backgrounds.

The GLINPUT Library Files.

The GLINPUT library consists of three main files:

- GLINPUTTOP.BI – a BASIC include file that needs to be located at the very top of your code.
- GLINPUT.BI – a BASIC include file that needs to be located at the very bottom of your code.
- GLINPUT_NOERROR.BI – a BASIC include file that needs to be located at the very bottom of your code. However, only use this file when you are sure that no errors exist in your code. GLINPUT.BI contains error trapping routines that help you to debug your program. GLINPUT_NOERROR.BI has had all of the error trapping routines stripped from it, allowing your final compiled project to run faster since these error checks are no longer needed.

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 are in your QB64 folder.

Constructing a Program

In the QB64 editor type in the following:

```
'$INCLUDE:'glinputtop.bi'

<a few blank lines>

'$INCLUDE:'glinput.bi'
```

GLINPUTTOP.BI contains the constant and type declarations needed for the library, and must always be at the top of your program's code. The file GLINPUT.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:

```
'$INCLUDE:'Libs\glinputtop.bi'
'$INCLUDE:'Libs\glinput.bi'
```

GLIINPUT() – defining an input field.

The library command `GLIINPUT` is used to define a graphics line input on screen. `GLIINPUT` has the following syntax:

handle% = GLIINPUT(*x%, y%, allowedtext%, displaytext$, savebackground%*)

Modify the code in your editor to look like this: (there is no need to capitalize the library commands; they will be auto-capitalized by the editor)

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
```

```
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '          show background image

helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)

'$INCLUDE:'glinput.bi'
```

The variable `helloworld%` is a handle, or pointer, that now contains a value that references this input. The input will not appear on the screen yet because all you've done is create an input field for later use. `100,100` points to the location on the graphics screen where the input will reside. `GLIALPHA` is a predefined constant that tells the input to allow only alphabetic input. `"Hello World: "` is the text that will precede the actual input field. `TRUE` tells the input field to maintain the integrity of the background image. Let's cover each of these options a little more in depth by starting with *`allowedtext%`*.

The following values can be used for *`allowedtext%`*:

Value	Predefined Constant	Meaning
1	`GLIALPHA`	Alphabetic characters allowed (A-Z, a-z)
2	`GLINUMERIC`	Numeric characters allowed (0-9)
4	`GLISYMBOLS`	Symbolic characters allowed (!@#$%^&* etc..)
8	`GLIDASH`	Dash (or minus) symbol allowed (-)
16	`GLIPAREN`	Parenthesis allowed ((and))
32	`GLILOWER`	Force input characters to lower case (a-z)
64	`GLIUPPER`	Force input characters to upper case (A-Z)
128	`GLIPASSWORD`	Mask input field with asterisks (*)

The values in the chart above can be used individually or combined to make custom allowances. For example, for most input fields you'll probably want to allow the ability for the user to type in any character. The value of 7, or `GLIALPHA + GLINUMERIC + GLISYMBOLS`, can be used to allow any keystroke to be used. However, if you create an input field for a phone number, the value of 26, or `GLINUMERIC + GLIDASH + GLIPAREN`, can be used to allow the user to input `(216)555-1212`. Adding 128, or `GLIPASSWORD`, will turn the input field into a password field masking any character typed in with an asterisk. `GLIUPPER` and `GLILOWER` will force the input field and resulting string to be upper or lower case.

The *`displaytext$`* parameter is optional but if used will display the string entered preceding the input field. If you wish to have no display text simply supply a null string (`""`).

The *`displaybackground%`* parameter is used to inform the input field whether or not it should preserve the background image underneath it. If set to `-1` (`TRUE`) the input field will maintain the background image by blending the text and if set to `0` (`FALSE`) a solid box will display underneath the input field. The background image will still be maintained however behind the solid input field. The color of the solid box will depend on the background color set with the `COLOR` command directly preceding the `GLIINPUT` command (more on that later).

GLICLEAR() – Removes all input fields from view and restores the background image.

The library command GLICLEAR is used to hide all of the input fields from view and restore the background image underneath each one. GLICLEAR has the following syntax:

```
GLICLEAR
```

GLICLEAR has no parameters. GLICLEAR must be the first command in any loop that contains input fields.

Modify the code in your editor to look like this: (from now on new lines will be **bold** to show which ones were added or changed)

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '         show background image

helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)
DO
    GLICLEAR ' must be first line in any loop


LOOP

'$INCLUDE:'glinput.bi'
```

GLIUPDATE() – Displays all active input fields on the screen.

Once you have created an input field(s) with GLIINPUT the GLIUPDATE command can be used to display the input fields to the screen. The library command has the following syntax:

```
GLIUPDATE
```

GLIUPDATE has no parameters and must be the second to last command in any loop, preceding the QB64 command _DISPLAY.

Modify the code in your editor to look like this:

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '         show background image

helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)
DO
    GLICLEAR '  must be first command in any loop

    ' <your code here>

    GLIUPDATE ' must be the second to last command in any loop
    _DISPLAY '  must be the last command in any loop to display results
LOOP

'$INCLUDE:'glinput.bi'
```

You now have the minimum code required for the library to function, but in the above example you'll get stuck in a never ending loop. There are a couple of different ways to test an input field and leave a loop based on the test.

GLIENTERED() – test one or all input fields for the ENTER key.

GLIENTERED tests one or all active input fields for the presence of the user having pressed the ENTER key. The syntax for the GLIENTERED command is as follows:

entered% = GLIENTERED(*handle%*)

The command will return -1 (TRUE) if the input field specified by *handle%* has had the ENTER key pressed or 0 (FALSE) if not. If the value of 0 (zero) is specified in *handle%* then GLIENTERED will return -1 (TRUE) only if all active input fields have had the ENTER key pressed on them.

Modify the code in your text editor again to see how GLIENTERED works.

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '         show background image

helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)
DO
    GLICLEAR '  must be first command in any loop

    ' <your code here>

    GLIUPDATE ' must be the second to last command in any loop
    _DISPLAY '  must be the last command in any loop to display results
LOOP UNTIL GLIENTERED(helloworld%)

'$INCLUDE:'glinput.bi'
```

At this point you have code that will exit the loop once the ENTER key has been pressed. Go ahead and execute the code. Type a few characters in (remember, only alphabetic characters will be accepted) and then use the left arrow key to go back a few characters and press the INSERT key. You'll notice the cursor changes from insert mode to overwrite mode. Pressing the DELETE key will delete the character to the right of the cursor and pressing the BACKSPACE key will delete the character to the left. The right arrow key will move the cursor to the right. The HOME key will move the cursor to the beginning of the input line and the END key will move the cursor to the end of the line. You can press the ENTER key no matter where the cursor is located to activate GLIENTERED.

GLIOUTPUT$() – retrieve the text from an input field.

Once your user has finished entering text you'll need to retrieve that text and this is where the GLIOUTPUT$ command comes in. The syntax for the GLIOUTPUT$ command is:

*text$* = GLIOUTPUT$(*handle%*)

GLIOUTPUT$ will retrieve the text from a GLIINPUT handle even while the user is typing into the input field. This allows the programmer to monitor what the user is typing in real time.

Modify your code once again to look like the code below:

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM helloworld$
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '         show background image

helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)
DO
    GLICLEAR '  must be first command in any loop

    LOCATE 1,1
    PRINT "Real time: "; GLIOUTPUT$(helloworld%); " "

    GLIUPDATE ' must be the second to last command in any loop
    _DISPLAY '  must be the last command in any loop to display results
LOOP UNTIL GLIENTERED(helloworld%)
helloworld$ = GLIOUTPUT$(helloworld%)
LOCATE 2, 1
PRINT "Final    : "; helloworld$

'$INCLUDE:'glinput.bi'
```

As you can see after executing the code above the GLIOUTPUT$ command can be used both inside your loops for real time monitoring and outside your loops to get the final text entered. But, in the above example, helloworld% is still active even after the loop has ended. When you are finished with an input field it must be closed to deactivate it.

GLICLOSE() – deactivates one or all input fields.

Once you are finished with one or all input fields you need to deactivate them. The syntax for the GLICLOSE command is:

GLICLOSE *handle%, behavior%*

GLICLOSE will deactivate the input field specified by *handle%*. If the value of 0 (zero) is passed through *handle%* then all active input fields will be closed at once. Setting a value of 0 (FALSE) for *behavior%* informs the GLICLOSE command to keep the input field visible on the screen and a value of -1 (TRUE) will remove the input field from the screen and restore the background image underneath.

Note that if you choose to have the input fields remain on the screen they will become part of the background image and GLIUPDATE will no longer maintain the background image for you.

Before we move onto the last few commands let's animate the background image we've been using to show off the capabilities of the GLINPUT library. Modify your code to the following:

```
'$INCLUDE:'glinputtop.bi'

CONST FALSE = 0, TRUE = NOT FALSE

DIM helloworld%
DIM helloworld$
DIM wallpaper&
DIM x%, y%

wallpaper& = _NEWIMAGE(640, 528, 32) ' create image to use as background
_DEST wallpaper&
LINE (0, 0)-(639, 527), _RGB32(0, 0, 127), BF
FOR y% = 1 TO 11
    FOR x% = 1 TO 13
        CIRCLE (x% * 48 - 17, y% * 48 - 24), 24, _RGB32(0, 0, 0)
        PAINT (x% * 48 - 17, y% * 48 - 24), _RGB32(0, 0, 96), _RGB32(0, 0, 0)
    NEXT x%
NEXT y%
SCREEN _NEWIMAGE(640, 480, 32)
_PUTIMAGE (0, 0), wallpaper& '         show background image
y% = 0
helloworld% = GLIINPUT(100, 100, GLIALPHA, "Hello World: ", TRUE)
DO
    GLICLEAR '  must be first command in any loop
    _LIMIT 32
    y% = y% - 1
    IF y% = -48 THEN y% = 0
    _PUTIMAGE (0, y%), wallpaper&
    LOCATE 1,1
    PRINT "Real time: "; GLIOUTPUT$(helloworld%); " "

    GLIUPDATE ' must be the second to last command in any loop
    _DISPLAY '  must be the last command in any loop to display results
LOOP UNTIL GLIENTERED(helloworld%)
helloworld$ = GLIOUTPUT$(helloworld%)
LOCATE 2, 1
PRINT "Final    : "; helloworld$
GLICLOSE helloworld%, TRUE

'$INCLUDE:'glinput.bi'
```

Go ahead and execute the code. As you can see the GLINPUT library will maintain background integrity whether you are using a static or dynamic background. We will not be using the above example program any longer, but there is no need to save it. The source file called 'glidocdemo1.bas' has been included with the library that contains the code above.

The remaining commands are going to require a bit more sophisticated example program. Included with this library is a source file called 'glidocdemo2.bas'. Go ahead and load this program now and execute it. Let's cover the remaining commands first and then investigate how they are being used in this demo program.

GLIFORCE() – force the cursor to either the next or specific input field.

GLIFORCE can be used to force an input field to become the new current field accepting input. The syntax for GLIFORCE is:

```
GLIFORCE handle%
```

Sending a value of -1 to GLIFORCE will move the cursor to the next active input field. The order of active input fields is the same order in which you created them with GLIINPUT. Sending a value greater than 0 (zero) forces the cursor to a specific active input field. In the demo program you just loaded you can see it is being used to force the cursor to active input fields that are required where no input was made.

GLICURRENT() – return the handle number of the active input field currently accepting input.

If you need to know where the user is currently located within the active input fields GLICURRENT can report this to you in the form of handle number. The syntax for GLICURRENT is as follows:

```
handle% = GLICURRENT
```

The example program you have loaded makes heavy use of GLICURRENT to know which information string to display to the user for the given active input the user is on, for instance. If the handle number passed back by GLICURRENT is 0 (zero) that means that there are no more active input fields present, in other words they have all been closed.

So far the examples programs have all made use of the default font for 32bit graphics screens in QB64, _FONT 16. But, the library is capable of using multiple fonts and font colors at the same time on the same screen. Included with the library are two font files that you'll need in your QB64 folder, 'lucon.ttf' and 'times.ttf', for the upcoming example.

Fonts and Colors

The library command GLIINPUT uses the current font and foreground/background colors that are set by the program. This allows you to change the font and color for each and every input on the screen if you wish. The basic outline for this is:

- Load your fonts using _LOADFONT somewhere at the beginning of your program
- Set the default font using _FONT
- Set the desired font foreground and background colors using COLOR and _RGB32
- Issue the GLIINPUT command and it will remember the font and colors previously set

- Repeat for each input created that needs a different font and/or color

Go ahead and load ‘glidocdemo3.bas’ now and execute it to see this in action.

Command Quick Reference

SUBROUTINES

GLICLOSE handle%, behavior% - deactivates one or all input fields.

GLIFORCE handle% - force the cursor to either the next or specific input field.

FUNCTIONS

handle% = GLIINPUT(x%, y%, allowedtext%, displaytext$, savebackground%) – define an input field.

text$ = GLIOUTPUT$(handle%) – retrieve the text from an input field.

entered% = GLIENTERED(handle%) – test one or all input fields for the ENTER key.

current% = GLICURRENT - return the handle number of the active input field currently accepting input.

CONSTANTS

Make sure not to use constants or variables with these names:

GLOBAL constants

GLIALPHA = 1
GLINUMERIC = 2
GLISYMBOLS = 4
GLIDASH = 8
GLIPAREN = 16
GLILOWER = 32
GLIUPPER = 64
GLIPASSWORD = 128

TYPE DECLARATIONS

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

TYPE GLI

ARRAYS

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

Gli() AS GLI