RITCHIE’S QB64 BUTTON LIBRARY

After having created yet another program that needed button input and writing the routines once again, I decided to make button creation easier on myself by writing a button library. Out of this need was born RITCHIE’S QB64 BUTTON LIBRARY. Having created the library I decided to share it with other QB64 programmers in hopes that they could not only benefit from it, but also help to improve it. I view this library as not only a way to solve a problem, but also as a means to better understand the inner workings of QB64.

This library is a set of commands created to manipulate buttons on a graphics screen. Handles are used to control and manipulate individual buttons on the screen. When creating the library the use of handles was emulated as much as possible in the same way that QB64 maintains handles for graphics, sounds, fonts, etc. This was done in hopes that the learning curve would be slight when using the library.

The library relies on no external variables or constants and the commands are completely non-destructive to the screen, meaning that placing, removing and moving buttons around the screen will always maintain the background image they were placed over. This frees the programmer from having to maintain variables or save screen segments before using any of the library’s commands.

LIMITATIONS in version 1.0

The library at this point is very simplistic. It will only support 32bit color screen modes and does not play well with Alpha transparency. You must either disable transparency before using the button commands, or use the CLS command after a new screen is created to remove transparency from the background. Buttons with transparency can still be used, however, by including a transparent layer in the Portable Network Graphics (PNG) files you create and load for custom buttons.

The library contains very basic error checking of handles only, ensuring that only valid handles are used between commands. Placing a button off screen, for instance, may have unpredictable and unforeseen results at this point.

The BUTTONNEW command will optionally create generic buttons from an internal button generator. This button generator is quaint at best and really needs the talents of a gifted graphics programmer to improve it. Because of this try to avoid passing any red, green or blue value to it that is below 64 or above 192. The method used to make the buttons look 3D is very crude and values outside this boundary are bound to make the 3D portion of the buttons look hideous. The library was written to take advantage of loading images for the buttons from the beginning. The custom button generator was simply added as a failsafe measure (see BUTTONNEW for details).

Included Files

- RQBLV10.PDF – This file you are currently viewing.
- RQBLTOP.BI – The BASIC include file that contains the variable declarations. This file is to be placed at the **TOP** of your source code.
- RQBL.BI – The BASIC include file that contains the library functions and procedures. This file is to be included at the **BOTTOM** of your source code.
- RQBLDEMO.BAS – A BASIC demo showing some of the uses of the library.
- Various .PNG files – The buttons and graphics used in the demo program and a few extra buttons to play with.
- lucon.ttf – The font used by the demo program.

Usage

There are two include files that must be included in the correct order and place. RQBLTOP.BI must be placed at the top of your source code. This include file contains the variable declarations needed for the library. RQBL.BI must be placed at the very bottom of your source code. This include file contains the function and subroutine declarations needed for the library. The outline below illustrates this:

```
‘$INCLUDE: ‘RQBLTOP.BI’
‘
‘<your CONST, TYPE and variable declarations>
‘
‘<your main program>
‘
‘<your FUNCTIONs and SUBroutines>
‘
‘$INCLUDE: ‘RQBL.BI’
```

BUTTONCHECKING (subroutine)

See also: BUTTONEVENT, BUTTONUPDATE

Purpose

The BUTTONCHECKING subroutine turns on or off button/mouse interaction auto-updating.

Syntax

`BUTTONCHECKING` *`state%`*

`state%` is the state to place automatic checking in. Valid values are:

-1 (true) - turn automatic checking on.
 0 (false) - turn automatic checking off.

Usage

```
CONST false = 0, true = NOT false ‘                                   set up boolean constants
CONST TURNOFF = false, TURNON = true ‘                                set up Boolean constants
SCREEN _NEWIMAGE(640, 480, 32) ‘                                      screen of 640x480 pixels
CLS ‘                                                                 remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button
BUTTONPUT 20, 20, mybutton% ‘                                         put button on screen
BUTTONCHECKING TURNON ‘                                               automatic event updates
DO ‘                                                                  begin looping
    LOCATE 1,1 ‘                                                      cursor upper left corner
    PRINT BUTTONEVENT(mybutton%) ‘                                    print button mouse status
LOOP UNTIL INKEY$ <> “” ‘                                             end loop when key struck
BUTTONCHECKING TURNOFF ‘                                              automatic updates off
BUTTONFREE mybutton% ‘                                                free button resources
```

The code above creates a new user screen of 640x480 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `20, 20`. Automatic button event checking is turned on before entering the `DO...LOOP`. The cursor is located at the upper left of the screen and the button’s current mouse interaction status is displayed on the screen. When a key is pressed automatic button event checking is turned off, the button freed from memory, and the button removed from the screen.

Description

The BUTTONCHECKING subroutine activates or deactivates an event timer that automatically updates mouse interactions for all buttons currently displayed on the screen. When BUTTONCHECKING is activated an event timer will poll the mouse/button relationship every 50 milliseconds (20 times per second). The event timer will continue to run until the command is issued to turn BUTTONCHECKING off. While automatic polling is turned on updates can be retrieved using the BUTTONEVENT function.

Tip

While BUTTONCHECKING is active your entire program will be forced to give up a few CPU cycles 20 times a second. If your program needs to know the status of all button and mouse interactions at all times then this is the best option. Another method would be to include BUTTONUPDATE inside of any loop that needs to retrieve button and mouse interaction data. BUTTONUPDATE updates all button events just the one time, freeing those lost CPU cycles.

BUTTONCOPY (function)

See also: BUTTONNEW

Purpose

The BUTTONCOPY function copies the button information from an existing handle to a new handle.

Syntax

newhandle% = BUTTONCOPY(*sourcehandle%*)

newhandle% is any integer variable that returns the handle number greater than zero (0) associated with the new button created by the copy procedure.

sourcehandle% is the button handle being copied.

Usage

```
DIM mybuttons%(10) ‘                                                     create button array
pushbutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ new temp button
FOR count% = 1 TO 10 ‘                                                   cycle through array
    mybuttons%(count%) = BUTTONCOPY(pushbutton%) ‘                       copy button to each element
NEXT count%
BUTTONFREE pushbutton% ‘                                                 release temp button
```

The code above illustrates how to create a button array using BUTTONCOPY. An integer array called mybuttons%() is created containing 10 elements. A button called pushbutton% is created to act as the source for BUTTONCOPY. A FOR...NEXT statement is used to copy pushbutton% into each element of the array. Finally, the source button that was created is freed now that it is not needed.

Description

The BUTTONCOPY function copies an existing handle’s information to a new handle. The newly created handle is treated just as if it were created with BUTTONNEW. All the characteristics of the source handle are copied to the new handle. However, if the source handle references a button that is currently shown on the screen, the new handle will not retain this characteristic. Other library routines will treat the new handle as having never been shown on the screen.

Tip

Freeing buttons that are no longer needed not only frees up resources, but also helps to improve the overall speed of the library.

BUTTONEVENT (function)

See also: BUTTONCHECKING, BUTTONUPDATE

Purpose

The BUTTONEVENT function returns the status of mouse interaction with a button.

Syntax

event% = BUTTONEVENT(*handle%*)

event% is any integer variable that returns the mouse interaction status of the button identified with *handle%*. The value returned will be one of the following four values:

0 - no current interaction with mouse pointer.
1 - left mouse button clicked down on button.
2 - right mouse button clicked down on button.
3 - mouse pointer hovering over button.

handle% is the button being tested for a mouse event.

xsize% and *ysize%* define the dimensional size of the button in pixels for the internal button generator. These values have no effect on button graphics defined by *fileset$*.

bcolor& defines the color of the button created by the internal button generator and has no effect on button graphics defined by *fileset$*.

Usage

```
CONST false = 0, true = NOT false ‘                                  set up boolean constants
CONST TURNOFF = false, TURNON = true ‘                               set up Boolean constants
SCREEN _NEWIMAGE(640, 480, 32) ‘                                     screen of 640x480 pixels
CLS ‘                                                                remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button
BUTTONPUT 20, 20, mybutton% ‘                                        put button on screen
BUTTONCHECKING TURNON ‘                                              automatic event updates
DO ‘                                                                 begin looping
    LOCATE 1,1 ‘                                                     cursor upper left corner
    PRINT BUTTONEVENT(mybutton%) ‘                                   print button mouse status
LOOP UNTIL INKEY$ <> “” ‘                                            end loop when key struck
BUTTONCHECKING TURNOFF ‘                                             automatic updates off
BUTTONFREE mybutton% ‘                                               free button resources
```

The code above creates a new user screen of 640x480 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `20, 20`. Automatic button event checking is turned on before entering the `DO...LOOP`. The cursor is located at the upper left of the screen and the button's current mouse interaction status is displayed on the screen. When a key is pressed automatic button event checking is turned off, the button freed from memory, and the button removed from the screen.

Description

The BUTTONEVENT function will return the current mouse interaction status of a button. However, BUTTONEVENT needs to be updated periodically to reflect the changes in status. This can be achieved with two methods. The first, as the program above highlights, is to turn automatic button event checking on through the use of BUTTONCHECKING. While BUTTONCHECKING is active the mouse interactions for every button showing on screen will be updated 20 times per second (50 milliseconds). Button interactions will continue to be monitored until BUTTONCHECKING is deactivated. The second method is to manually update the mouse interactions by using the BUTTONUPDATE command. Calling BUTTONUPDATE will update all mouse interactions for every button showing on screen just one time.

Tip

While BUTTONCHECKING is active your entire program will be forced to give up a few CPU cycles 20 times a second. If your program needs to know the status of all button and mouse interactions at all times then this is the best option. Another method would be to include BUTTONUPDATE inside of any loop that needs to retrieve button and mouse interaction data. BUTTONUPDATE updates all button events just the one time, freeing those lost CPU cycles.

BUTTONEXISTS (function)

See also: BUTTONNEW

Purpose

The BUTTONEXISTS function tests for the presence of the two Portable Network Graphics (PNG) files needed to create a graphics button set.

Syntax

`valid%` = BUTTONEXISTS(*`fileset$`*)

`valid%` will contain zero (0) (false) if the button set does not exist or -1 (true) if it does.

`fileset$` is the core name of the files that contain the button graphics.

Usage

```
buttonpresent% = BUTTONEXISTS("pushbutton") '         check for existence of button graphics
IF buttonpresent% THEN '                              does it exist?
    PRINT "The graphics button set exists." '         yes, tell user it exists
ELSE '                                                no
    PRINT "The graphics button set does not exist." ' tell user it does not exist
END IF
```

The code above sets the integer variable `buttonpresent%` to zero (0) (false) or -1 (true) by using `BUTTONEXISTS` to test for the existence of the graphics button set `"pushbutton"`. The program then reports to the user the result of the test.

Description

The BUTTONEXISTS function searches for the two files needed to create a graphics button set. A value of -1 (true) will be returned if the button set exists and a value of zero (0) (false) will be returned if it is not present. The button graphics, specified by *`fileset$`*, will be verified to exist. The button graphics must be in Portable Network Graphics (PNG) format and there must be two present. Both graphic files must start with the same name, "pushbutton" for instance. The button graphic that contains the image of the button in a depressed state (off or out) must end in zero (0), and the image that contains the button in a pressed state (on or in) needs to end in one (1). Therefore, a valid file set for "pushbutton" would be "pushbutton0.png" and "pushbutton1.png". When referencing this file set using the BUTTONEXISTS function only the common name of the file set is used, in this case "pushbutton".

Tip

The BUTTONNEW function uses BUTTONEXISTS to verify the presence of a graphics button set before creating a new button handle. Using BUTTONEXISTS before BUTTONNEW would be redundant.

BUTTONFREE (subroutine)

See also: BUTTONNEW, BUTTONHIDE

Purpose

The BUTTONFREE subroutine removes a button and frees any resources it is using.

Syntax

BUTTONFREE *handle%*

handle% is a button handle that was previously created with either BUTTONNEW or BUTTONCOPY.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) ‘                                  create 800x600 screen
CLS ‘                                                             remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button handle
BUTTONPUT 100, 100, mybutton% ‘                                   place button on screen
SLEEP ‘                                                           wait for key press
BUTTONFREE mybutton% ‘                                            free button from memory
END ‘                                                             terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. After the user presses a key the button’s resources are freed from memory, which also removes it from the screen.

Description

The BUTTONFREE subroutine will remove a button that was created with either BUTTONNEW or BUTTONCOPY. The button will also be removed from the screen and the background image restored if the button is showing.

Tip

If you simply want to remove a button from the screen, without freeing its resources, you can use the BUTTONHIDE command instead.

BUTTONHEIGHT (function)

See also: BUTTONWIDTH, BUTTONNEW

Purpose

The BUTTONHEIGHT function returns the integer height of a button in pixels.

Syntax

bheight% = BUTTONHEIGHT(*handle%*)

bheight% will contain the integer height of the button identified by *handle%*.

handle% is the handle number referencing the button to retrieve the height from.

Usage

```
mybutton% = BUTTONNEW("pushbutton", 64, 32, _RGB(127, 127, 127)) ' create new button
bheight% = BUTTONHEIGHT(mybutton%) '                               get the button's height
PRINT "The width of the button is"; bwidth% '                      inform the user
```

The code above creates a new button called *mybutton%* then gets the height of that button, storing the value in *bheight%*. The result is then reported to the user.

Description

The BUTTONHEIGHT function returns the height of a button specified by *handle%*. If a valid button graphics set was loaded through BUTTONNEW the height value returned will be the height of the button set. If an invalid, or null string, was supplied to BUTTONNEW the height value will be the value identified in *ysize%* of BUTTONNEW.

BUTTONHIDE (subroutine)

See also: BUTTONON, BUTTONOFF, BUTTONFREE, BUTTONSHOW, BUTTONTOGGLE

Purpose

The BUTTONHIDE subroutine hides, or makes invisible, a button currently showing on screen.

Syntax

BUTTONHIDE *handle%*

handle% is the handle of the button you wish to hide or make invisible.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) '                                  create 800x600 screen
CLS '                                                             remove alpha transparency
mybutton% = BUTTONNEW("pushbutton", 64, 32, _RGB(127, 127, 127)) ' create new button handle
BUTTONPUT 100, 100, mybutton% '                                   place button on screen
SLEEP '                                                           wait for key press
BUTTONHIDE mybutton% '                                            hide button from view
SLEEP '                                                           wait for key press
BUTTONFREE mybutton% '                                            free button from memory
END '                                                             terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the CLS command to remove the alpha transparency channel. A new handle called mybutton% is created that points to a button graphic set called "pushbutton". If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by mybutton% is then placed on the screen at coordinates 100, 100. When the user presses a key the button is hidden from view using BUTTONHIDE. After the user presses a key the button's resources are freed from memory, which also removes it from the screen.

Description

The BUTTONHIDE subroutine hides, or makes invisible, the button specified by the handle given.

Tip

BUTTONFREE will also hide a button by removing it from the screen and memory. Use BUTTONHIDE if you wish to reuse the same button later on without having to recreate it with BUTTONNEW.

BUTTONNEW (function)

See also: BUTTONFREE

Purpose

The BUTTONNEW function creates a new button handle either from a set of files containing the button graphics or from an internal generic button generator.

Syntax

handle% = BUTTONNEW(*fileset$*, *xsize%*, *ysize%*, *bcolor&*)

handle% is any integer variable that returns the handle number greater than zero (0) associated with the new button.

fileset$ is the core name of the files that contain the button graphics. If set to null, or the graphic images do not exist, an internal button generator will create a button set from *xsize%*, *ysize%* and *bcolor&*.

xsize% and *ysize%* define the dimensional size of the button in pixels for the internal button generator. These values have no effect on button graphics defined by *fileset$*.

bcolor& defines the color of the button created by the internal button generator and has no effect on button graphics defined by *fileset$*. (See Known Issues below)

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) ‘                                    create 800x600 screen
CLS ‘                                                               remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button handle
BUTTONPUT 100, 100, mybutton% ‘                                     place button on screen
SLEEP ‘                                                             wait for key press
BUTTONFREE mybutton% ‘                                              free button from memory
END ‘                                                               terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. After the user presses a key the button’s resources are freed from memory, which also removes it from the screen.

Description

The BUTTONNEW function is used to create a handle that points to a button graphic that can be manipulated on screen. The button graphics, specified by *fileset$*, will be loaded if they are present. The button graphics must be in Portable Network Graphics (PNG) format and there must be two present. Both graphic files must start with the same name, “pushbutton” for instance. The button graphic that contains the image of the button in a depressed state (off or out) must end in zero (0), and the image that contains the button in a pressed state (on or in) needs to end in one (1). Therefore, a valid file set for “pushbutton” would be “pushbutton0.png” and “pushbutton1.png”. When referencing this file set using the BUTTONNEW function only the common name of the file set is used, in this case “pushbutton”. PNG files that contain a transparency layer are also supported by BUTTONNEW. If an invalid file set is selected, or a null string (“”) is entered, the BUTTONNEW function will create a generic button based off of the information contained in *xsize%*, *ysize%* and *bcolor&*.

Tip

Even if you have a valid graphics file set it may be a good idea to specify *xsize%*, *ysize%* and *bcolor&*. You can think of this as a “failsafe” measure that will kick in if for any reason your button graphics become corrupted or unavailable. If this happens, a generic button will take the place of your button graphic until you can discover why the graphic set is not loading. Also, by specifying *xsize%*, *ysize%* and *bcolor&*, you don’t have to concentrate on creating your button graphics first. You can instead concentrate on the code you are creating, leaving the creation of the button graphics for later.

Known Issues

The custom button creation routine in the button library has issues with red, green or blue values above 192 and below 64. Try to keep your red, green and blue color values between 64 and 192. The routine that creates the 3D look of the button adds and subtracts 64 from the red, green and blue values to give the button its highlighted look toward the upper left and faded look in the lower right. The BUTTONNEW’s custom button generation routine could really use the talents of a good graphics programmer to go in and spruce it up. If you redo this routine please send me a copy of your changes and they will be noted in the official documentation along with credit given to you.

BUTTONOFF (subroutine)

See also: BUTTONON, BUTTONHIDE, BUTTONSHOW, BUTTONTOGGLE

Purpose

The BUTTONOFF subroutine places a button into a depressed, or off, state.

Syntax

`BUTTONOFF` *`handle%`*

`handle%` is the handle of the button you wish to place in a depressed, or off, state.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) ‘                                    create 800x600 screen
CLS ‘                                                               remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button handle
BUTTONON mybutton% ‘                                                turn button on
BUTTONPUT 100, 100, mybutton% ‘                                     place button on screen
SLEEP ‘                                                             wait for key press
BUTTONOFF mybutton% ‘                                               turn button back off
SLEEP ‘                                                             wait for key press
BUTTONFREE mybutton% ‘                                              free button from memory
END ‘                                                               terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button is placed into a pressed state using `BUTTONON`. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. When the user presses a key the button is placed back into a depressed, or off, state with the use of `BUTTONOFF`. After the user presses a key the button’s resources are freed from memory, which also removes it from the screen.

Description

The BUTTONOFF subroutine places a button into a depressed, or off, state. The change will also be reflected on screen if the button is currently showing.

Tip

BUTTONOFF can be used on buttons that are not visible as well.

BUTTONON (subroutine)

See also: BUTTONOFF, BUTTONHIDE, BUTTONSHOW, BUTTONTOGGLE

Purpose

The BUTTONON subroutine places a button into a pressed, or on, state.

Syntax

BUTTONON *handle%*

handle% is the handle of the button you wish to place in a pressed, or on, state.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) '                                   create 800x600 screen
CLS '                                                              remove alpha transparency
mybutton% = BUTTONNEW("pushbutton", 64, 32, _RGB(127, 127, 127)) ' create new button handle
BUTTONON mybutton% '                                               turn button on
BUTTONPUT 100, 100, mybutton% '                                    place button on screen
SLEEP '                                                            wait for key press
BUTTONFREE mybutton% '                                             free button from memory
END '                                                              terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the CLS command to remove the alpha transparency channel. A new handle called mybutton% is created that points to a button graphic set called "pushbutton". If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button is placed into a pressed state using BUTTONON. The button specified by mybutton% is then placed on the screen at coordinates 100, 100. After the user presses a key the button's resources are freed from memory, which also removes it from the screen.

Description

The BUTTONON subroutine places a button into a pressed, or on, state. The change will also be reflected on screen if the button is currently showing.

Tip

BUTTONON can be used on buttons that are not visible as well. The first time BUTTONPUT is used on a newly created button it defaults to a depressed, or off, state. You can use BUTTONON before BUTTONPUT if you wish to have the button show up in a pressed, of on, state first.

BUTTONPUT (subroutine)

See also: BUTTONON, BUTTONOFF, BUTTONHIDE, BUTTONSHOW

Purpose

The BUTTONPUT subroutine places a button the screen or moves a button to a new screen location.

Syntax

BUTTONPUT *x%*, *y%*, *handle%*

x% and *y%* are the coordinates on the screen to either place the button for the first time or move the button to a new location.

handle% is the handle of the button to place or move.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) ‘                                  create 800x600 screen
CLS ‘                                                             remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button handle
BUTTONPUT 100, 100, mybutton% ‘                                   place button on screen
SLEEP ‘                                                           wait for key press
BUTTONPUT 200, 200, mybutton% ‘                                   move button to new location
SLEEP ‘                                                           wait for key press
BUTTONFREE mybutton% ‘                                            free button from memory
END ‘                                                             terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. When the user presses a key the original background image of where the button resides is restored and the button is moved to coordinates `200, 200`. After the user presses a key again the button’s resources are freed from memory, which also removes it from the screen.

Description

The BUTTONPUT subroutine is used to place a button on the screen for the first time or to move a button that has been previously placed on the screen with BUTTONPUT. When a button is placed on the screen for the first time a copy of the screen area where the button will reside is saved. Any button command that manipulates the button on screen will then use this background image to restore the original screen.

BUTTONSHOW (subroutine)

See also: BUTTONON, BUTTONOFF, BUTTONHIDE, BUTTONTOGGLE

Purpose

The BUTTONSHOW subroutine makes visible on screen a button previously hidden by BUTTONHIDE.

Syntax

BUTTONSHOW *handle%*

handle% is the handle of the button you wish to make visible.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) ‘                                   create 800x600 screen
CLS ‘                                                              remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button handle
BUTTONPUT 100, 100, mybutton% ‘                                    place button on screen
SLEEP ‘                                                            wait for key press
BUTTONHIDE mybutton% ‘                                             hide button from view
BUTTONPUT 200, 200, mybutton% ‘                                    move while invisible
SLEEP ‘                                                            wait for key press
BUTTONSHOW mybutton% ‘                                             show button in new spot
SLEEP ‘                                                            wait for key press
BUTTONFREE mybutton% ‘                                             free button from memory
END ‘                                                              terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. When the user presses a key the button is hidden from view using `BUTTONHIDE` and moved to coordinates `200, 200` while invisible using `BUTTONPUT`. When the user presses a key the button is once again shown in its new spot using `BUTTONSHOW`. After the user presses a key the button’s resources are freed from memory, which also removes it from the screen.

Description

The BUTTONSHOW subroutine makes visible on screen a button previously hidden by BUTTONHIDE.

BUTTONTOGGLE (subroutine)

See also: BUTTONON, BUTTONOFF, BUTTONHIDE, BUTTONSHOW

Purpose

The BUTTONTOGGLE subroutine alternately toggles the state of the button between pressed and depressed (on and off).

Syntax

`BUTTONTOGGLE` *`handle%`*

`handle%` is the handle of the button you wish to toggle the state of.

Usage

```
SCREEN _NEWIMAGE(800, 600, 32) '                                   create 800x600 screen
CLS '                                                              remove alpha transparency
mybutton% = BUTTONNEW("pushbutton", 64, 32, _RGB(127, 127, 127)) ' create new button handle
BUTTONPUT 100, 100, mybutton% '                                    place button on screen
FOR count% = 1 TO 10 '                                             10 iterations
    BUTTONTOGGLE mybutton% '                                       toggle button's state
    _DELAY .5 '                                                    wait 1/2 second
NEXT count%
BUTTONFREE mybutton% '                                             free button from memory
END '                                                              terminate program
```

The code above creates a new user screen of 800x600 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `"pushbutton"`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates `100, 100`. The button is then toggled ten times over a five second period, producing an animation effect on screen. The button's resources are then freed from memory, which also removes it from the screen.

Description

The BUTTONTOGGLE subroutine flip-flops between the two button graphics available, a pressed state and depressed state. Calling BUTTONTOGGLE on a button that is currently pressed, or on, will cause the button to go into a depressed, or off, state. Likewise, calling BUTTONTOGGLE on a button that is currently depressed, or off, will cause the button to go into a pressed, or on, state. The change will also be reflected on screen if the button is showing.

Tip

BUTTONTOGGLE can be used on buttons that are not visible as well. Newly created, or hidden buttons, can have their state toggled before showing them on screen.

BUTTONUPDATE (subroutine)

See also: BUTTONEVENT, BUTTONCHECKING

Purpose

The BUTTONUPDATE subroutine polls all current on-screen buttons for their mouse interactions.

Syntax

BUTTONUPDATE

Usage

```
SCREEN _NEWIMAGE(640, 480, 32) ‘                                   screen of 640x480 pixels
CLS ‘                                                              remove alpha transparency
mybutton% = BUTTONNEW(“pushbutton”, 64, 32, _RGB(127, 127, 127)) ‘ create new button
BUTTONPUT 20, 20, mybutton% ‘                                      put button on screen
DO ‘                                                               begin looping
    BUTTONUPDATE ‘                                                 poll button/mouse interaction
    LOCATE 1,1 ‘                                                   cursor upper left corner
    PRINT BUTTONEVENT(mybutton%) ‘                                 print button mouse status
LOOP UNTIL INKEY$ <> “” ‘                                          end loop when key struck
BUTTONFREE mybutton% ‘                                             free button resources
```

The code above creates a new user screen of 640x480 pixel dimension and utilizes the `CLS` command to remove the alpha transparency channel. A new handle called `mybutton%` is created that points to a button graphic set called `“pushbutton”`. If this file set is not present then the internal button generator will create a button of 64x32 pixel dimension and gray in color. The button specified by `mybutton%` is then placed on the screen at coordinates 20, 20. `BUTTONUPDATE` is called to get the current interaction between the mouse and `mybutton%`. The cursor is located at the upper left of the screen and the button’s current mouse interaction status is displayed on the screen. When a key is pressed the button is freed from memory and the button removed from the screen.

Description

The BUTTONUPDATE subroutine polls all currently showing buttons for their current mouse interaction status. The updates can be retrieved using the BUTTONEVENT function.

Tip

While BUTTONCHECKING is active your entire program will be forced to give up a few CPU cycles 20 times a second. If your program needs to know the status of all button and mouse interactions at all times then this is the best option. This method using BUTTONUPDATE inside of any loop that needs to retrieve button and mouse interaction data is done only when called, freeing those lost CPU cycles.

BUTTONWIDTH (function)

See also: BUTTONHEIGHT, BUTTONNEW

Purpose

The BUTTONWIDTH function returns the integer width of a button in pixels.

Syntax

`*bwidth%*` = BUTTONWIDTH(*handle%*)

bwidth% will contain the integer width of the button identified by *handle%*.

handle% is the handle number referencing the button to retrieve the width from.

Usage

```
mybutton% = BUTTONNEW("pushbutton", 64, 32, _RGB(127, 127, 127)) ' create new button
bwidth% = BUTTONWIDTH(mybutton%) '                                 get the button's width
PRINT "The width of the button is"; bwidth% '                      inform the user
```

The code above creates a new button called *mybutton%* then gets the width of that button, storing the value in *bwidth%*. The result is then reported to the user.

Description

The BUTTONWIDTH function returns the width of a button specified by *handle%*. If a valid button graphics set was loaded through BUTTONNEW the width value returned will be the width of the button set. If an invalid, or null string, was supplied to BUTTONNEW the width value will be the value identified in *xsize%* of BUTTONNEW.