The GUI Software Toolkit for the QScreen Controller

Kernel Verson 4.4

Glossary of GUI Functions

www.mosaic-industries.com

April 2004

RELEASE 3/31/2004 KERNEL EXTENSION V4.4

Glossary of Terms **5**
Action flag 5
Array_pf struct 5
Button location 5
Button object 6
Col, Row 7
Dualmode 7
Graphic object 7
Graphics array 7
Keymap array 8
Menu 8
Menu manager 9
Modified xaddress 9
Object handler 9
Text array 9
Tvars struct 10
Xaddress 12

Glossary of Functions **13**
Forth: ADD_BUTTON (button_location\ col\ row\ action_mask\ button_obj --) 16
C: ADD_BUTTON(uint button_location, uint col, uint row, uint action_mask, BUTTON * button_obj) 16
Forth: ADD_GRAPHIC(col\ row\ action_mask\ graphic_obj_xaddr\ --) 16
C: ADD_GRAPHIC(uint col, uint row, uint action_mask, xaddr graphic_obj_xaddr) 16
Forth: ADD_TOUCH_BUTTON(col\ row\ action_mask\ button_obj --) 17
C: ADD_TOUCH_BUTTON(uint col, uint row, uint action_mask, BUTTON * button_obj) 17
Forth: BLANKBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr\ press_graphic_xaddr\ press_handler\ release_handler\ label1\ label2\ label3\ label4 <name> --) 18
C: BLANKBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr, xaddr press_graphic_xaddr, (void *) press_handler, (void *) release_handler, char * label1, char * label2, char * label3, char * label4,<name>) 18
Forth: BUILD_MENU 18
C: BUILD_MENU(arrayname, num_elements) 18
Forth: Button_Draw (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Draw(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19
Forth: Button_Draw_Textonly (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Draw_Textonly(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19
Forth: Button_Erase_Textonly (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Erase_Textonly(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19
Forth: Button_Press (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Press(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19
Forth: Button_Release (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Release(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19
Forth: Button_Repeat (col\ row\ tvars_addr\ tvars_page\ xpfa --) 19
C: void Button_Repeat(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr) 19

Forth: Calibrate_Touchscreen (raw0\raw1\raw2 --) 19
C: void Calibrate_Touchscreen(long raw0, long raw1, long raw2) 19
Forth: Clear_Graphics (tvars_addr\ tvars_page --) 20
C: void Clear_Graphics(GUI_VARS * tvars_addr, page tvars_page) 20
Forth: Clear_Pixel (x \ y --) 20
C: void Clear_Pixel(uint x, uint y) 20
Clears a pixel directly from the LCD display bypassing the graphics array. Since it erases directly from the screen, and not from the graphics array, calling Update_Graphics will rewrite anything removed from the screen by Clear_Pixel. See Set_Pixel. 20
Forth: Clear_Text (tvars_addr\ tvars_page --) 20
C: void Clear_Text(GUI_VARS * tvars_addr, page tvars_page) 20
Forth: colrow_to_button (row\ col -- button number) 20
C: COLROW_TO_BUTTON(col,row) 20
Forth: Config_Display (graphics_cols\ graphics_rows\ graphics_start\ background_fill\ text_cols\ text_rows\ text_start\ heap_bottom\ heap_top\ tvars_addr\ tvars_page --) 20
C: void Config_Display(uint graphics_cols, uint graphics_rows, addr graphics_start, uchar background_fill, uint text_cols, uint text_rows, addr text_start, xaddr heap_bottom, xaddr heap_top, GUI_VARS * tvars_addr, page tvars_page) 20
Forth: Direct_Draw_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --) 21
C: void Direct_Draw_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr) 21
Forth: Direct_Erase_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --) 21
C: void Direct_Erase_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr) 21
Forth: Do_Button (col\ row\ tvars_addr\ tvars_page\ action\ xpfa --) 21
C: void Do_Button(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, uint action, BUTTON * button_xaddr) 21
Forth: Do_Graphic (col\ row\ tvars_addr\ tvars_page\ action\ graphic_xaddr --) 23
C: void Do_Graphic (uint col, int row, GUI_VARS * tvars_addr, page tvars_page, uint action, FORTH_CONST_ARRAY * graphic_xaddr) 23
Forth: Do_Menu (col\ row\ tvars_addr\ tvars_page\ action\ menu_xaddr --) 24
C: void Do_Menu(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, uint action, MENU * menu_xaddr) 24
Forth: Draw_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --) 24
C: void Draw_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr) 24
Forth: Erase_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --) 24
C: void Erase_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr) 24
Forth: FASTBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr \press_graphic_xaddr \ handler\ label1\ label2\ label3\ label4 <name> --) 24
C: FASTBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr xaddr press_graphic_xaddr, (void *) handler, char * label1, char * label2, char * label3, char * label4,<name>) 24
Forth: Init_Display (tvars_addr\ tvars_page --) 25
C: void Init_Display(GUI_VARS * tvars_addr, page tvars_page) 25
Forth: Init_Menu (offset\ col\ row\ tvars_addr\ tvars_page\ menu xpfa --) 25
C: void Init_Menu(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr) 25
Forth: Init_Touch (tvars_addr\ tvars_page --) 25
C: void Init_Touch(GUI_VARS * tvars_addr, page tvars_page) 25
Forth: Menu_Install (offset\ col\ row\ tvars_addr\ tvars_page\ menu_xaddr --) 25
C: void Menu_Install(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr) 25
Forth: Menu_Remove (offset\ tvars_addr\ tvars_page\ menu_addr\ menu_page --) 26
C: void Menu_Remove(uint offset, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr) 26

Forth: NEW_MENU: and BUILD_MENU 26
C: NEW_MENU and BUILD_MENU(arrayname, num_elements) 26
Forth: NORMBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr \press_graphic_xaddr \ handler\ label1\ label2\ label3\ label4 <name> --) 27
C: NORMBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr, xaddr press_graphic_xaddr, (void *) handler, char * label1, char * label2, char * label3, char * label4,<name>) 27
Forth: Read_Touchscreen (tvars_addr \ tvars_page – n | 0 <= n <= 20) 28
C: int Read_Touchscreen(GUI_VARS * tvars_addr, page tvars_page) 28
Forth: Read_Raw_Coords (tvars_addr \ tvars_page – raw_coords) 28
C: long Read_Raw_Coords(GUI_VARS * tvars_addr, page tvars_page) 28
Forth: Service_Touch (button number \ tvars_addr \ tvars_page --) 28
C: void Service_Touch(int button, GUI_VARS * tvars_addr, page tvars_page) 28
Forth: Set_Cursor_State (isvisible\ isflashing --) 29
C: void Set_Cursor_State(boolean isvisible, boolean isflashing) 29
Forth: Set_Display_Mode (mode --) 29
C: void Set_Display_Mode(uint mode) 29
Forth: Set_Display_State (graphics\ text --) 29
C: void Set_Display_State(boolean graphics, boolean text) 29
Forth: Set_Gr_Area (columns --) 29
C: void Set_Gr_Area(uint columns) 29
Forth: Set_Gr_Home_Addr (address --) 29
C: void Set_Gr_Home_Addr(addr address) 29
Forth: Set_Pixel (x\y --) 30
C: void Set_Pixel(uint x, uint y) 30
Forth: Set_Text_Area (columns --) 30
C: void Set_Text_Area(uint columns) 30
Forth: Set_Text_Home_Addr (address --) 30
C: void Set_Text_Home_Addr(addr address) 30
Forth: Set_Text_Mode (modebyte --) 30
C: void Set_Text_Mode(uchar modebyte) 30
Forth: Std_Display (tvars_addr\ tvars_page --) 31
C: void Std_Display(GUI_VARS * tvars_addr, page tvars_page) 31
Forth: Uninit_Menu (offset\ col\ row\ tvars_addr\ tvars_page\ menu xpfa --) 31
C: void Uninit_Menu(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr) 31
Forth: Update_Graphics (tvars_addr\ tvars_page --) 31
C: void Update_Graphics (GUI_VARS * tvars_addr, page tvars_page) 31
Forth: Update_Here_With (address\ graphics_resource_addr\ graphics_resource_page\ garray_xaddr --) 32
C: void Update_Here_With(addr address, addr * graphics_resource_addr, page graphics_resource_page, FORTH_ARRAY * garray_xaddr) 32
Forth: Update_Text (tvars_addr\ tvars_page --) 32
C: void Update_Text(GUI_VARS * tvars_addr, page tvars_page) 32
Forth: Update_Text_And_Graphics (tvars_addr\ tvars_page --) 32
C: void Update_Text_And_Graphics(GUI_VARS * tvars_addr, page tvars_page) 32
Forth: Wait_For_Press (tvars_addr\ tvars_page --) 32
C: void Wait_For_Press(GUI_VARS * tvars_addr, page tvars_page) 32
Forth: Wait_For_Release (tvars_addr\ tvars_page --) 32
C: void Wait_For_Release(GUI_VARS * tvars_addr, page tvars_page) 32
Forth: Wait_Then_Service_Touch (tvars_addr\ tvars_page --) 32
C: void Wait_Then_Service_Touch(GUI_VARS * tvars_addr, page tvars_page) 32

Glossary of Terms

Action flag

A flag that is passed to an object handler with an object to determine the behavior of the object. Some actions are used by several object types and others have meaning only to one specific type of object. The actions are described in the Glossary of Functions under the object handlers for which they are used. See Do_Graphic and Do_Button for contextual explanations of the actions.

Array_pf struct

Forth array parameter field structure describing the geometry of a forth array. The actual array data is contained in the heap pointed to by a heap handle.

CFORTH_ARRAY structure			
Offset	**Type**	**Element name**	**Description**
0x0000	xaddr	handle	Xaddress of the pointer (heap xhandle) to the beginning of the data
0x0004	addr	cur_heap	16 bit addr of top of heap (same page as xhandle)
0x0006	uint	bytes_per_element	# of bytes in each element
0x0008	uint	num_dimensions	# of dimensions in array
0x000A	uint	num_cols	# of columns (1st dimension)
0x000C	uint	num_rows	# of rows (2nd dimension)
0x000E	uint	num_pages	# of pages (3rd dimension)
0x0010	uint	num_books	# of books (4th dimension)

Button location

An integer value describing one of the 20 touch sensitive areas of the touchscreen panel. This quantity is used to index the keymap array when buttons are pressed. Below is a map of the button locations on the touchscreen.

1	5	9	13	17
2	6	10	14	18
3	7	11	15	19
4	8	12	16	20

Button object

A data structure of type BUTTON that describes a button. The button object handler, Do_Button, accepts the xaddress of this structure, an action flag, and a col/row location.

BUTTON structure			
Offset	**Type**	**Element name**	**Description**
0x0000	uint	flags	A set of bitmapped flags describing button's behavior
0x0002	xaddr	graphic_handler_xcfa	The xaddress of Do_Graphic
0x0006	xaddr	draw_graphic	Xaddress of graphic object drawn for DRAW_ACTION
0x000A	xaddr	release_graphic	Xaddress of graphic object drawn for RELEASE_ACTION
0x000E	xaddr	press_graphic	Xaddress of graphic object drawn for PRESS_ACTION
0x0012	xaddr	press_handler	Xaddress of user code executed for PRESS_ACTION
0x0016	xaddr	release_handler	Xaddress of user code executed for RELEASE_ACTION
0x001A	xaddr	label1	Modified xaddress of text string for line 1 of the button
0x001E	xaddr	label2	Modified xaddress of text string for line 2 of the button
0x0022	xaddr	label3	Modified xaddress of text string for line 3 of the button
0x0026	xaddr	label4	Modified xaddress of text string for line 4 of the button

The flags in the above structure determine the specific behavior of the button. All other elements are irrelevant if unused. The flags, which are constants that may be ORed together, determine which elements will be required. Button defining macros configure common flags. If a bit is set, then the flag is true. If it is cleared, then it is false. The possible flags are:

DRAW_GRAPHIC_FLAG

Indicates that there is a valid xaddress for a graphic object in the draw_graphic field to be drawn for the DRAW_ACTION.

RELEASE_GRAPHIC_FLAG

Indicates that there is a valid xaddress for a graphic object in the release_graphic field to be drawn for the RELEASE_ACTION.

PRESS_GRAPHIC_FLAG

Indicates that there is a valid xaddress for a graphic object in the press_graphic field to be drawn for the PRESS_ACTION.

DIR_DRAW_GRAPHIC_FLAG

When drawing the draw_graphic object, use direct to screen drawing.

DIR_RELEASE_GRAPHIC_FLAG

When drawing the release_graphic object, use direct to screen drawing.

DIR_PRESS_GRAPHIC_FLAG

When drawing the press_graphic object, use direct to screen drawing.

DRAW_TEXT_FLAG
Print the text strings whose xaddress are stored in label1, label2, label3, and label4 in the button.

PRESS_HANDLER_FLAG
Execute the code at the xaddress stored in the press_handler field when the button is given a PRESS_ACTION.

RELEASE_HANDLER_FLAG
Execute the code at the xaddress stored in the release_handler field when the button is given a RELEASE_ACTION.

REPEAT_FLAG
If the button is pressed and held, start executing the press_handler repeatedly. If the press handler is not enabled by having the PRESS_HANDLER_FLAG set, then the REPEAT_FLAG will have no effect.

TEXT_UPDATE_PRESS_FLAG
Call Update_Text when the button is given a PRESS_ACTION.

TEXT_UPDATE_RELEASE_FLAG
Call Update_Text when the button is given a RELEASE_ACTION.

GRAPHICS_UPDATE_PRESS_FLAG
Call Update_Graphics when the button is given a PRESS_ACTION.

GRAPHICS_UPDATE_RELEASE_FLAG
Call Update_Graphics when the button is given a RELEASE_ACTION.

C_STYLE_TEXT_FLAG
Interpret strings in label fields as C style strings instead of Forth.

The macros used to create buttons are **FASTBUTTON**, **NORMBUTTON**, and **BLANKBUTTON**. They are described in the *Glossary of Functions*.

Col, Row
Used to describe position on the LCD screen. When used in reference to graphics, unless otherwise stated, col is a unit of 6 horizontal pixels and row is a unit of 1 vertical pixel. When used in reference to text, unless otherwise stated, col is a unit of 1 character width, 6 pixels, and row is a unit of 1 character line, 8 pixels.

Dualmode
Refers to the technique of operating the LCD display simultaneously in text and graphics modes. The QScreen has internal display routines that control the text layer of the display while the dualmode driver extensions contained in the GUI Toolkit extend the capabilities to allow graphics mode to be used in tandem with text mode.

Graphic object
Image to be displayed on the LCD display handled by Do_Graphic. Graphic objects are constant 2 dimensional forth arrays generated by the Image Conversion Program. The Image Conversion Program also generates a header file comprising constants that refer to the xaddresses of the graphic objects. See Do_Graphic.

Graphics array
An array in RAM that contains a shadow of the graphics layer on the display. The graphics array is a forth array dimensioned in the display heap area. The tvars struct

contains the graphics array's parameter field (array_pf struct). The location of the display heap is also specified in the tvars struct. DRAW_ACTION, REDRAW_ACTION, or ERASE_ACTION cause objects to write data to this array. Subsequently calling Update_Graphics sends the graphics array data to the display. Direct screen writes completely bypass this array. If Update_Graphics is called after a direct screen write, then any data not mirrored in the graphics array is overwritten. Init_Display dimensions and fills the graphics array with the background_fill member of tvars.

Keymap array

A forth array whose parameter field is stored in the tvars struct. This is an array of structures of type KEYMAP_ENTRY. It has the number of elements equal to the number of touch sensitive areas on the touchscreen or keypad. For the standard QScreen controller there are 20 elements. When a the touchscreen area is pressed or released, the corresponding element of the keymap array is examined to determine if anything should done in response. The menu manager looks to see if that element contains a valid button object, and if so the button object's handler is executed. The parameters passed to the handler are the location of the button in the menu, the button object's xaddress, and the PRESS_ACTION, RELEASE_ACTION, or REPEAT_ACTION. If that element contains 0x00000000 in the object field, then the button press and the subsequent release are ignored.

KEYMAP_ENTRY structure			
Offset	**Type**	**Element name**	**Description**
0x0000	uint	row	The relative or absolute row position for the button graphics*
0x0002	uint	col	The relative or absolute col position for the button graphics*
0x0004	xaddr	object	The xaddress of the object structure
0x0008	xaddr	obj_handler	The xaddress of the object handler

* When this structure is used as part of the keymap, the row and col are absolute. When it is used as part of a menu (see MENU_ENTRY struct), the row and col are relative displacements from the upper left corner of the menu.

Menu

An array of structures, each of type MENU_ENTRY, that serves as a grouping of button and graphic objects. This array also contains location information about each object. A typical menu might contain several buttons and perhaps a logo or other graphic objects. Multiple menus may be displayed at the same time. Conflicts will occur if the same location is used by two menus at the same time however. Each element of the array is of type MENU_ENTRY. Macros simplify the creation of menus. See NEW_MENU, ADD_BUTTON, ADD_GRAPHIC, and BUILD_MENU for information on creating menus. See Init_Menu, Uninit_Menu, Do_Menu for information on using menus. Each element of a menu is of the following form:

MENU_ENTRY structure			
Offset	**Type**	**Element name**	**Description**
0x0000	KEYMAP_ENTRY	keymap_entry	A sub-structure of type keymap_entry that will be copied into the keymap array when the menu is installed.
0x000C	uint	action_mask	Bitmask used to control what action flags may be passed through to the object for Do_Menu.
0x000E	uint	button	The relative keymap index of a button object. If a nonzero number is passed to Init_Menu for the button offset, then it is added to the relative keymap index to form an absolute keymap index.

Menu manager

The software routine that scans the keypad or touchscreen hardware and makes calls to objects stored in the keymap array. The standard menu manager used in the GUI Toolkit is called Wait_Then_Service_Touch, but any routine that can collect user input and make the appropriate calls to the objects stored in a menu can act as a menu manager.

Modified xaddress

See xaddress.

Object handler

The function that processes an object and its parameters and initiates its behavior. Do_Graphic and Do_Button are the object handlers for graphic and button objects respectively. These object handlers may be called from C as described in the Glossary of Functions section, but they are also required as part of certain structures such as BUTTON and KEYMAP_ENTRY. When programming in C, use the following syntax for placing the 24 bit xaddress of these handlers in the structures when building them manually.

TO_XADDR(DO_GRAPHIC_ADDR, DO_GRAPHIC_PAGE)
TO_XADDR(DO_BUTTON_ADDR, DO_BUTTON_PAGE)

The macros that assist in button and menu creation eliminate the need to build those structures manually. Consequently, there is rarely a need to explicitly specify the xaddresses. See Do_Graphic and Do_Button.

Text array

An array in RAM that contains a shadow of the text layer on the display. The text array is a forth array dimensioned in the display heap area. Its parameter is field stored at the address location returned by GARRAY_XPFA which is always in common RAM. The forth equivalent garray.xpfa returns an xaddress with 0x00 as the page. Although for historical reasons the name implies that it is used for

graphics, the GUI Toolkit only uses this array for storing text. This location is the standard location for the display array parameter field on the QScreen. The elements of this array contain ASCII data that has been shifted down by 0x20. The Toshiba TC6963 chip on the display uses this modified ASCII method for storing text. StringToDisplay (forth: $>display) requires that the text array to be located here. Init_Display dimensions and clears the text array by filling it with 0x00 (ASCII space shifted by 0x20).

Tvars struct

A structure that contains all the global variables used by the GUI Toolkit. The user's program must contain an instance of this structure of type GUI_VARS. Throughout this document and in all of the example routines, the instance of this structure is named tvars. The macro, TVARS is a replacement for (tvars, THIS_PAGE). Many of the GUI Toolkit's functions require the base address of this structure as one of the arguments. Below is a description of the elements of this structure.

GUI_VARS structure			
Offset	**Type**	**Element name**	**Description**
0x0000	xaddr	display_heap_top	Xaddress of top (last byte) of display heap
0x0004	xaddr	display_heap_bottom	Xaddress of bottom (first byte) of display heap
0x0008	int	background_fill	Background fill byte for graphic layer. The lower 6 bits describe a 6 pixel horizontal field.
0x000A	array_pf	graphics_garray	18 byte long array_pf sub-structure for the graphics layer array. This array is dimensioned and initialized in the display heap area by Init_Display.
0x001C	xaddr	display_resource	Display resource variable for access control
0x0020	addr	gr_home_addr	Address of graphics area on the display controller
0x0022	addr	text_home_addr	Address of text area on the display controller
0x0024	uint	graphic_rows	# of pixel lines on the display
0x0026	uint	graphic_cols	# of cols on the display (6 pixels/byte) 1 col=6 pixels
0x0028	uint	text_rows	# of text lines on the display
0x002A	uint	text_cols	# of text columns on the display
0x002C	array_pf	keymap_array	18 byte long array_pf sub-structure for the keymap array. This array_pf struct is initialized and the array is dimensioned and initialized to 0xFF by Init_Touch in the current heap when Init_Touch is called.
0x003E	uint	current_row*	The row of the button being pressed
0x0040	uint	current_col*	The col of the button being pressed
0x0042	uint	current_keynum*	The button number of the button being pressed
0x0044	xaddr	current_button*	The xaddress of the button object being pressed
0x0048	uint	repeat_delay	# of timeslicer counts to wait before repeating
0x004A	uint	repeat_period	# of timeslicer counts to wait between repetitions

* These quantities are never read by the GUI Toolkit routines. They are written to by the menu manager to provide a means for the user's handler code to implement location sensitive behavior. For example, a handler could use these variables to implement a modal set of selector buttons so that when one button is pressed, it stays in the 'pressed' position while releasing the previously pressed button.

Xaddress

A 24 bit number consisting of a 16 bit address and an 8 bit page. Xaddresses occupy 32 bit fields. C functions that use pointers only return 16 bit addresses. The macros used in the GUI Toolkit pad out the 16 bit addresses as needed to accommodate the functions that do require full xaddresses. A modified xaddress is used to describe strings of two possible types, forth style and C style. If the upper 8 bits of the xaddress of a string are set to 0xFF, then the xaddress will be interpreted as a C style null terminated string. If the upper 8 bits are set to 0x00, then the string will be interpreted as a forth style counted string.

Glossary of Functions

Many of the functions in this glossary are not needed for typical GUI based applications. They are provided to allow advanced programmers to use the tools in a more manual and flexible way. These are the functions that are likely to be used in any application.

ADD_BUTTON	Adds button to menu in a menu definition
ADD_GRAPHIC	Adds graphic to menu in a menu definition
ADD_TOUCH_BUTTON	Adds touchscreen button to a menu
BUILD_MENU	Terminates the creation of a menu
Calibrate_Touchscreen	Calibrate the touchscreen
Clear_Graphics	Clears the graphics layer
Clear_Pixel	Clear a pixel from the display
Clear_Text	Clears the text layer
Do_Graphic	Handles graphic object actions
FASTBUTTON	Creates a new button that is fast drawing
Init_Display	Initializes the LCD display hardware
Init_Menu	Displays and activates a menu
Init_Touch	Initializes touchscreen environment vars
NEW_MENU	Begins a new menu definition
Read_Touchscreen	Read the calibrated touchscreen
Read_Raw_Coords	Read raw values from the touchscreen
Service_Touch	Processes a button press passed to it
Set_Cursor_State	Sets flashing and visibility cursor attributes
Set_Pixel	Set a pixel on the display
Std_Display	Configures the LCD for typical defaults
Uninit_Menu	Erases a menu from screen and nullifies it
Update_Graphics	Updates the graphics layer on the LCD
Update_Text	Updates the text layer on the LCD
Update_Text_And_Graphics	Updates text and graphics layers
Wait_For_Press	Wait for press of the touchscreen before returning
Wait_For_Release	Wait for release of the touchscreen before returning
Wait_Then_Service_Touch	Polls the touchscreen for a press

#include <mosaic/gui_tk/to_large.h> Must specify this before each set of button or menu definitions if programming in C

#include <mosaic/gui_tk/fr_large.h> Must specify this at the end of each set of button or menu definitions if programming in C

The Forth and C header files have many constants defined in them. Constants shown in italics are only included in the Forth header file. Most of these are not useful in most applications, but are provided to allow advanced programmers to have better access to the GUI Toolkit configuration information. A complete listing of the constants in use by the GUI Toolkit is provided below. If any of these names is used in the user's Forth code, a non-unique warning will be issued. In C programs, a preprocessor error is issued. The constants in this list that are generally useful are described in detail in their relevant glossary entries and manual sections.

Display hardware constants

**PRIOR_CURSOR_STATE*	Address of a display driver control variable
GRAPHICS_DATA_ADDR	Hardware address of display data port
GRAPHICS_CMD_ADDR	Hardware address of display command port
AWSET_CMD	Command for AutoWrite Set
AWRESET_CMD	Command for AutoWrite Release
SET_TX_HOME_CMD	Command to set text layer home display address
SET_TX_AREA_CMD	Command to set text layer width
SET_GR_HOME_CMD	Command to set graphics layer home address
SET_GR_AREA_CMD	Command to set graphics layer width
MODE_CMD	Command to set display controller mode
DISPLAY_MODE_CMD	Command to set display visibility mode
CURSOR_BLINK	Bit flag to set cursor blink state
CURSOR_ON	Bit flag to set cursor visibility
TEXT_MODE	Bit flag to enable/disable text mode
GRAPHICS_MODE	Bit flag to enable/disable graphics mode
OR_TEXT	Bit flag to set text to be ORed with graphics
EXOR_TEXT	Bit flag to set text to be XORed with graphics
AND_TEXT	Bit flag to set text to be ANDed with graphics
LCD_TEXT_ADDR	Default text home layer display address
LCD_GRAPHIC_ADDR	Default graphics layer home display address
LINES_PER_CHAR	Number of pixel rows per text mode character
GRAPHICS_COLUMNS	Number of columns in bytes (6 pixels)
GRAPHICS_ROWS	Number of pixel rows on the graphics layer
TEXT_COLUMNS	Number of text columns on text layer
TEXT_ROWS	Number of lines of text on text layer
DISPLAY_HEAP_TOP	Xaddress of display heap top
DISPLAY_HEAP_BOTTOM	Xaddress of display heap bottom
DEF_BACKGROUND_FILL	Default fill byte for graphic layer background

Graphic object action flags

GRAPHICS_MASK	Action mask for graphic objects
DRAW_ACTION	Draw action flag
DIR_DRAW_ACTION	Direct Draw action flag
REDRAW_ACTION	Redraw action flag
ERASE_ACTION	Erase action flag
DIR_ERASE_ACTION	Direct erase action flag

Button object action flags

BUTTON_MASK	Action mask for button objects
PRESS_REPEAT_ACTION	Press while repeating action flag
PRESS_ACTION	Press action flag
REL_ACTION	Release action flag
DRAW_TEXTONLY_ACTION	Draw only text portion action flag
ERASE_TEXTONLY_ACTION	Erase only text portion action flag

Button object configuration flags

DRAW_GRAPHIC_FLAG	Button uses draw graphic
RELEASE_GRAPHIC_FLAG	Button uses release graphic
PRESS_GRAPHIC_FLAG	Button uses press graphic
DIR_DRAW_GRAPHIC_FLAG	Button directly draws draw graphic
DIR_RELEASE_GRAPHIC_FLAG	Button directly draws release graphic
DIR_PRESS_GRAPHIC_FLAG	Button directly draws press graphic

DRAW_TEXT_FLAG	Button has text labels
PRESS_HANDLER_FLAG	Button has a press handler
RELEASE_HANDLER_FLAG	Button has a release handler
REPEAT_FLAG	Button is repeating
TEXT_UPDATE_PRESS_FLAG	Update_Text called when button is pressed
TEXT_UPDATE_RELEASE_FLAG	Update_Text called when button is released
GRAPHICS_UPDATE_PRESS_FLAG	Update_Graphics called when button is pressed
GRAPHICS_UPDATE_RELEASE_FLAG	Update_Graphics called when button is released
C_STYLE_TEXT_FLAG	Use C style interpretation of label strings

Menu object action flags

MENU_MASK	Action mask for menu objects (obsolete)
INIT_ACTION	Init action flag (obsolete)
UNINIT_ACTION	Uninit action flag (obsolete)

Menu object entry configuration flags

BUTTON_NULL	Disable menu entry slot
BUTTON_NONLOCAL	Disable entry insertion into keymap
DEFAULT_REPEAT_PERIOD	Default repeat period for repeating buttons
DEFAULT_REPEAT_DELAY	Default repeat delay for repeating buttons

Constants used by the object building macros

BUTTON_COLS	Number of button columns on the touchscreen
BUTTON_ROWS	Number of button rows on the touchscreen
BUTTON_WIDTH	Number of columns comprising a button
BUTTON_HEIGHT	Number of pixel rows comprising a button
GRAPHICS_MASK	Action mask for all drawing actions
FASTBUTTON_FLAGS	Button configuration flags for fastbuttons
NORMBUTTON_FLAGS	Button configuration flags for normbuttons

Forth: ADD_BUTTON (button_location\ col\ row\ action_mask\ button_obj --)
C: ADD_BUTTON(uint button_location, uint col, uint row, uint action_mask, BUTTON * button_obj)

Adds a button object to a menu. Inside a menu definition, ADD_BUTTON is a macro that inserts all the necessary information for a button object based on the relative screen position specified by col and row and the address of the graphic object. The button_location describes the relative button number used for this button. If you are adding a touchscreen button, the macro ADD_TOUCH_BUTTON may be used to eliminate the need for specifying the button_location. ADD_TOUCH_BUTTON computes the button_location automatically based on the col and row given. See Init_Menu for more details on how the button number is used. The action_mask is a mask used to control which actions may be passed to the object when Do_Menu is called. For most applications, DRAW_MASK should be used. All of the actions are single bit flags thus several actions can be ORed together to form an action mask. DRAW_MASK is a constant that ORs DRAW_ACTION, ERASE_ACTION, DIR_DRAW_ACTION, REDRAW_ACTION, DIR_ERASE_ACTION, DRAW_TEXTONLY_ACTION, and ERASE_TEXTONLY_ACTION. Here is an example of the usage:

Forth:

```
NEW_MENU: mymenu_menu
...
12  3  32  DRAW_MASK  mybutton1  ADD_BUTTON
...
...
BUILD_MENU
```

C:

```
NEW_MENU mymenu[4]=
{
... ,
ADD_BUTTON(12,  3, 32, DRAW_MASK, mybutton1),
... ,
...
};
BUILD_MENU( mymenu, 4);
```

Also see the example in the glossary entry for NEW_MENU.

Forth: ADD_GRAPHIC(col\ row\ action_mask\ graphic_obj_xaddr\ --)
C: ADD_GRAPHIC(uint col, uint row, uint action_mask, xaddr graphic_obj_xaddr)

Adds a graphic object to a menu. Inside a menu definition, ADD_GRAPHIC is a macro that inserts all the necessary information for a graphic object based on the relative screen position specified by col and row and the address of the graphic object. The action_mask is a mask used to control which actions may be passed to the object when Do_Menu is called. For most applications, DRAW_MASK should be used. All of the actions are single bit flags thus several actions can be ORed together to form an action mask. DRAW_MASK is a constant that ORs DRAW_ACTION, ERASE_ACTION, DIR_DRAW_ACTION, REDRAW_ACTION, DIR_ERASE_ACTION, DRAW_TEXTONLY_ACTION, and

ERASE_TEXTONLY_ACTION. Since a graphic is simply a static image, it has no touchscreen button associated with it. Here is an example of the usage:

Forth:

```
NEW_MENU: mymenu_menu
...
3  32  DRAW_MASK     MY_LOGO  ADD_GRAPHIC
...
...
BUILD_MENU
```

C:

```
NEW_MENU mymenu[4]=
{
... ,
ADD_GRAPHIC( 3, 32,  DRAW_MASK,  MY_LOGO),
... ,
...
};
BUILD_MENU( mymenu, 4);
```

Also see the example in the glossary entry for NEW_MENU.

Forth: ADD_TOUCH_BUTTON(col\ row\ action_mask\ button_obj --)
C: ADD_TOUCH_BUTTON(uint col, uint row, uint action_mask, BUTTON * button_obj)

Places an initializing data into a menu for a button. Inside a menu definition, ADD_TOUCH_BUTTON is a macro that inserts all the necessary information for a button object based on the relative screen position specified by col and row and the address of the graphic object. The button_location is computed automatically based on the col and row given since there is a direct relationship between the touchscreen and the col and row position. The action_mask is a mask used to control which actions may be passed to the object when Do_Menu is called. For most applications, DRAW_MASK should be used. All of the actions are single bit flags thus several actions can be ORed together to form an action mask. DRAW_MASK is a constant that ORs DRAW_ACTION, ERASE_ACTION, DIR_DRAW_ACTION, REDRAW_ACTION, DIR_ERASE_ACTION, DRAW_TEXTONLY_ACTION, ERASE_TEXTONLY_ACTION. Here is an example of the usage:

Forth:

```
NEW_MENU: mymenu_menu
...
3  32  DRAW_MASK  mybutton1  ADD_TOUCH_BUTTON
...
...
BUILD_MENU
```

C:

```
NEW_MENU mymenu[4]=
{
... ,
ADD_TOUCH_BUTTON(3, 32, DRAW_MASK, mybutton1),
... ,
...
};
BUILD_MENU( mymenu, 4);
```

Also see the example in the glossary entry for NEW_MENU.

Forth: BLANKBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr\ press_graphic_xaddr\ press_handler\ release_handler\ label1\ label2\ label3\ label4 <name> --)

C: BLANKBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr, xaddr press_graphic_xaddr, (void *) press_handler, (void *) release_handler, char * label1, char * label2, char * label3, char * label4,<name>)

Creates a new button object. This macro integrates the creation and initialization of the BUTTON structure. BLANKBUTTON is a lower level macro than its more used cousins, FASTBUTTON and NORMBUTTON. It simply automates the creation of the object. The specified value for the flags is stored in the flags field of the button. The other parameters are stored in their respective fields in new button. The name given to the new button is specified by <name>. Here is an example:

Forth:

```
DRAW_GRAPHIC_FLAG                \ It has a draw graphic
RELEASE_GRAPHIC_FLAG or          \ It has a release graphic
PRESS_GRAPHIC_FLAG or            \ It has a press graphic
DRAW_TEXT_FLAG or                \ It has text
PRESS_HANDLER_FLAG or            \ It has a press handler
GRAPHICS_UPDATE_PRESS_FLAG or    \ Call Update_Graphics on press
GRAPHICS_UPDATE_RELEASE_FLAG or  \ Call Update_Graphics on release
LBLANK_PCX                       \ Graphic for DRAW_ACTION
LBLANK_PCX                       \ Graphic for RELEASE_ACTION
LBLACK_PCX                       \ Graphic for PRESS_ACTION
cfa.for myfunction        \ The code xaddress for press handler
0\0                              \ Dummy value for release handler
" "                              \ Line 1 label
" Start"                        \ Line 2 label
" Pump"                         \ Line 3 label
" "                              \ Line 4 label
BLANKBUTTON mybutton1            \ Instantiate the new button
```

C:

```
BLANKBUTTON(
DRAW_GRAPHIC_FLAG |        // Has draw graphic
RELEASE_GRAPHIC_FLAG |           // It has a release graphic
PRESS_GRAPHIC_FLAG |             // It has a press graphic
DRAW_TEXT_FLAG |                 // It has text
PRESS_HANDLER_FLAG |             // It has a press handler
GRAPHICS_UPDATE_PRESS_FLAG |     // Call Update_Graphics on press
GRAPHICS_UPDATE_RELEASE_FLAG,    // Call Update_Graphics on release
LBLANK_PCX,                      // Graphic for DRAW_ACTION
LBLANK_PCX,                      // Graphic for RELEASE_ACTION
LBLACK_PCX,                      // Graphic for PRESS_ACTION
myfunction,                      // The code address for press handler
0,                               // Dummy value for release handler
"",                              // Line 1 label
"Start",                         // Line 2 label
"Pump",                          // Line 3 label
"",                              // Line 4 label
mybutton1);                      // Instantiate the new button
```

Forth: BUILD_MENU

C: BUILD_MENU(arrayname, num_elements)

See NEW_MENU.

Forth: Button_Draw (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Draw(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the DRAW_ACTION of a button. See **Do_Button.**

Forth: Button_Draw_Textonly (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Draw_Textonly(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the DRAW_TEXTONLY_ACTION of a button. See **Do_Button.**

Forth: Button_Erase_Textonly (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Erase_Textonly(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the ERASE_TEXTONLY_ACTION of a button. See **Do_Button.**

Forth: Button_Press (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Press(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the PRESS_ACTION of a button. See **Do_Button.**

Forth: Button_Release (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Release(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the RELEASE_ACTION of a button. See **Do_Button.**

Forth: Button_Repeat (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Button_Repeat(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, BUTTON * button_xaddr)
Direct call to that carries out the REPEAT_ACTION of a button. See **Do_Button.**

Forth: Calibrate_Touchscreen (raw0\raw1\raw2 --)
C: void Calibrate_Touchscreen(long raw0, long raw1, long raw2)
A function that calibrates the analog touchscreen using the raw touchscreen readings of three points. The raw touchscreen readings are obtained with Read_Raw_Coords. The points are chosen to avoid non-linearities (points that are not too close to the edge), minimize scaling errors (points that are not too close to each other), and yield non-redundant simultaneous equations. The raw touchscreen readings are turned into coefficients that are applied to raw touchscreen readings each time the touchscreen is pressed. The coefficients are stored into flash.

Forth: Clear_Graphics (tvars_addr\ tvars_page --)
C: void Clear_Graphics(GUI_VARS * tvars_addr, page tvars_page)
Clears the graphics array by filling it with background_fill, a member of the tvars struct. Clear_Graphics then calls Update_Graphics so that the graphics layer on the display is cleared.

Forth: Clear_Pixel (x \ y --)
C: void Clear_Pixel(uint x, uint y)
Clears a pixel directly from the LCD display bypassing the graphics array. Since it erases directly from the screen, and not from the graphics array, calling Update_Graphics will rewrite anything removed from the screen by Clear_Pixel. See Set_Pixel.

Forth: Clear_Text (tvars_addr\ tvars_page --)
C: void Clear_Text(GUI_VARS * tvars_addr, page tvars_page)
Clears the text array by filling it with spaces. ASCII values in this array are shifted down by 0x20, a requirement of the TC6963 display controller. Clear_Text then calls Update_Text so that the text layer on the display is cleared.

Forth: colrow_to_button (row\ col -- button number)
C: COLROW_TO_BUTTON(col,row)
Converts from column and row coordinates to a button number (0-19). This macro is used by ADD_TOUCH_BUTTON to convert the specified graphical positional information to a button number corresponding to the touchscreen button location.

Forth: Config_Display (graphics_cols\ graphics_rows\ graphics_start\ background_fill\ text_cols\ text_rows\ text_start\ heap_bottom\ heap_top\ tvars_addr\ tvars_page --)
C: void Config_Display(uint graphics_cols, uint graphics_rows, addr graphics_start, uchar background_fill, uint text_cols, uint text_rows, addr text_start, xaddr heap_bottom, xaddr heap_top, GUI_VARS * tvars_addr, page tvars_page)
Fills the variables that control the display initialization with configuration parameters. Below is a summary of the parameters.

graphic_cols, graphic_rows -- the col, row size of the graphics array
graphics_start -- the starting address inside the LCD display for the graphics data
background_fill -- the background fill byte used by the Clear_Graphics function
text_cols, text_rows -- the col, row size of the text array
text_start -- the starting address inside the display for the text data
heap_bottom -- the xaddress of the first byte of the display heap to be used
heap_top -- the xaddress of the last byte of the display heap to be used
tvars_addr, tvars_page -- the address of the structure tvars

Although this function fills the variables that control the initialization of the display, it does NOT initialize the display. That must be done by Init_Display. Config_Display simply initializes the variables needed by Init_Display. Std_Display calls this function to set up the display according to a generic set of defaults. If you are using a QScreen Controller, use Std_Display. See Std_Display for the default values.

Forth: Direct_Draw_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Direct_Draw_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr)
Direct call to that carries out the DIR_DRAW_ACTION of a graphic. See **Do_Graphic.**

Forth: Direct_Erase_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Direct_Erase_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr)
Direct call to that carries out the DIR_ERASE_ACTION of a graphic. See **Do_Graphic.**

Forth: Do_Button (col\ row\ tvars_addr\ tvars_page\ action\ xpfa --)
C: void Do_Button(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, uint action, BUTTON * button_xaddr)
Action handler for all button objects. Col and row specify the location of the upper left corner of the image. Tvars_addr and tvars_page is the global structure that contains control variables used by the GUI Toolkit. This function's behavior is determined by the action passed to it. The actions are described below. The button_xaddr refers to a button object's xaddress. The button objects are structures of type BUTTON that contain xaddresses of three graphic objects, the draw graphic, release graphic, and press graphic. The button structure also contains a bitmapped set of switches that further shape the behavior of the button as well as xaddresses for the user code to be executed upon press, release, or both depending on which bitmapped flags are set. Here is how Do_Button is called:

```
C: Do_Button ( <col>, <row>, <tvars>, <action>, <button_xaddr> );
Forth: <col> <row> <tvars> <action> <button_xaddr> Do_Button
```

Buttons are used as parts of menus. The menu manager is responsible for calling this function to react to a touchscreen or keypad button press/release detection. Generally, your application would not call this function directly, but you can use this function to simulate button a press/release. This function can be used to produce the exact same effect as actually pressing or releasing the button. The action passed to Do_Button is one of the following predefined constants:

DRAW_ACTION
Draws the draw graphic member of the button structure to the graphics array with col, row as the upper left corner. You must subsequently call Update_Graphics to

make the image appear on the LCD display unless the DIR_DRAW_GRAPHIC_FLAG is set in which case the graphic will be drawn directly to the display. See DIR_DRAW_ACTION under Do_Graphic.

DIR_DRAW_ACTION
For button objects, this action is equivalent to DRAW_ACTION. In order for a button to be directly drawn to the display, the flag DIR_DRAW_GRAPHIC_FLAG must be set. See DRAW_ACTION.

REDRAW_ACTION
For button objects, does the same thing as DRAW_ACTION.

ERASE_ACTION
Writes the tvars background_fill byte to the graphics array in the area previously occupied by the button object at specified screen location. If the DIR_DRAW_GRAPHIC_FLAG is set, then the button is directly erased from the display. If that flag is not set, then you must then call Update_Graphics to make the change evident on the screen.

DIR_ERASE_ACTION
For button objects, this action is equivalent to ERASE_ACTION. In order for a button to be directly erased from the display, the flag DIR_DRAW_GRAPHIC_FLAG must be set. See ERASE_ACTION.

DRAW_TEXTONLY_ACTION
If the button has text labels, then this action causes them to be printed to the display. You must call Update_Text for this to become apparent on the screen.

ERASE_TEXTONLY_ACTION
If the button has text labels, then this action causes them to be erased from the display. You must call Update_Text for this to become apparent on the screen.

PRESS_ACTION
Executes the user code press_handler and draws the release graphic depending on the value of the flags. If the PRESS_HANDLER_FLAG is set, then the code in the press_handler field of the button structure is executed. If the PRESS_GRAPHIC_FLAG is set, then the graphic object for the press_graphic field is drawn. If the TEXT_UPDATE_PRESS_FLAG is set, then Update_Text is called. If the GRAPHIC_UPDATE_PRESS_FLAG is set, then Update_Graphics is called. If none of those flags is set, then this action has no effect.

RELEASE_ACTION
Executes the user code release_handler and draws the release graphic depending on the value of the flags. If the RELEASE_HANDLER_FLAG is set, then the code in the release_handler field of the button structure is executed. If the RELEASE_GRAPHIC_FLAG is set, then the graphic object for the release_graphic field is drawn. If the TEXT_UPDATE_RELEASE_FLAG is set, then Update_Text is called. If the GRAPHIC_UPDATE_RELEASE_FLAG is set, then Update_Graphics is called. If none of those flags is set, then this action has no effect.

REPEAT_ACTION
Executes the user code press_handler without drawing the press graphic. If the REPEAT_FLAG or PRESS_HANDLER flags are not set, this action has no

effect. When a button is repeating, it is a waste of processor time to redraw the same graphic for each repetition.

Forth: Do_Graphic (col\ row\ tvars_addr\ tvars_page\ action\ graphic_xaddr --)
C: void Do_Graphic (uint col, int row, GUI_VARS * tvars_addr, page tvars_page, uint action, FORTH_CONST_ARRAY * graphic_xaddr)

Action handler for all graphics objects. Col and row describe the position of the upper left corner of the object in absolute coordinates measured from the upper left corner of the display. This function's behavior is determined by the action flag passed to it. The actions are described below. The graphic_xaddr refers to a graphics object. Typically, such objects are created by the Image Conversion Program which converts a pcx or bmp graphic image on a PC to a block of data that can be loaded into the QScreen. The Image Conversion Program also provides a symbol listing of constants named based on the filename of the graphic on the PC. This symbol listing may be #included in a C file, or pasted into a forth file. The language in which the constants are defined can be set using the Advanced Dialog Box of the Image Conversion Program. The constant referring to the graphic objects address takes the place of the graphic_xaddr. For example, if the original image was named logo.pcx on the PC, then its name as a graphic object would be LOGO_PCX. Do_Graphic would then be called as follows:

```
C:  Do_Graphic ( <col>, <row>, <tvars>, <action>, LOGO_PCX );
Forth: <col> <row> <tvars> <action> LOGO_PCX Do_Graphic
```

The actions passed to Do_Graphic can be one of the following predefined constants:

DRAW_ACTION
Draws the graphic object to the graphics array with col, row as the upper left corner. You must subsequently call Update_Graphics to make the image appear on the LCD display.

DIR_DRAW_ACTION
Draws the graphics object directly to the LCD display bypassing the graphics array. This is useful for fast screen updates and animation. Since DIR_DRAW_ACTION draws directly to the screen, and not to the graphics array, calling Update_Graphics will overwrite anything placed on the screen by DIR_DRAW_ACTION.

REDRAW_ACTION
For graphic objects, does the same thing as DRAW_ACTION.

ERASE_ACTION
Writes the tvars background_fill byte to the graphics array in the area previously occupied by the graphic object at specified screen location. You must then call Update_Graphics to make the change evident on the screen.

DIR_ERASE_ACTION
Writes the tvars background_fill byte directly to the LCD display over the area previously occupied by the graphic object at specified screen location. See DIR_DRAW_ACTION.

Forth: Do_Menu (col\ row\ tvars_addr\ tvars_page\ action\ menu_xaddr --)
C: void Do_Menu(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, uint action, MENU * menu_xaddr)
Counts through each element of the menu executing each object with the specified action. A menu may consist of graphic or button objects. For example, calling Do_Menu with the action DRAW_ACTION would execute each object in the menu with that action. The relative col and row of the objects is stored in the menu. The col and row passed to Do_Menu is the desired position on the display of the upper left corner of the entire menu. The col and row passed to Do_Menu will be added to the col and row of the objects stored in the menu to get the absolute locations of the actual objects contained in the menu. That new col and row will then be passed to the object along with the action flag. Most commonly, this function is used to draw or redraw a menu to the screen. Each element of the menu array has an action mask which is ANDed with the action flag passed to Do_Menu before the object in the menu is executed. If the ANDed result is zero, the object is skipped. This allows certain objects in a menu to have some action flags disabled. See the following example:

```
C:  Do_Menu ( <col>, <row>, <tvars>, <action>, <menu_xaddr> );
Forth: <col> <row> <tvars> <action> <menu_xaddr> Do_Menu
```

Forth: Draw_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Draw_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr)
Direct call to that carries out the DRAW_ACTION of a graphic. See **Do_Graphic.**

Forth: Erase_Graphic (col\ row\ tvars_addr\ tvars_page\ xpfa --)
C: void Erase_Graphic(uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, FORTH_CONST_ARRAY * graphic_xaddr)
Direct call to that carries out the ERASE_ACTION of a graphic. See **Do_Graphic.**

Forth: FASTBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr \press_graphic_xaddr \ handler\ label1\ label2\ label3\ label4 <name> --)
C: FASTBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr xaddr press_graphic_xaddr, (void *) handler, char * label1, char * label2, char * label3, char * label4,<name>)
Works in exactly the same way as NORMBUTTON, but with a different set of default flags. FASTBUTTON has the default flags, DRAW_GRAPHIC_FLAG, RELEASE_GRAPHIC_FLAG, PRESS_GRAPHIC_FLAG, DIR_PRESS_GRAPHIC_FLAG, and DIR_RELEASE_GRAPHIC_FLAG. This type of button uses the direct screen drawing for the pressed and released graphics, and standard graphics array drawing for the initial drawing of the buttons. This technique is quite effective for maximizing the responsiveness of the user interface while still loosely following the paradigm of using a graphics array. It eliminates the need to update the entire screen when only a small portion the size of a button is changing. When using direct to screen drawing, you should not specify the GRAPHICS_UPDATE_PRESS_FLAG or

GRAPHICS_UPDATE_RELEASE_FLAG since updating the display will overwrite the directly drawn graphics. See **graphics objects** in the *Glossary of Terms* for more information. Also see **NORMBUTTON** in the *Glossary of Functions*.

Forth: Init_Display (tvars_addr\ tvars_page --)
C: void Init_Display(GUI_VARS * tvars_addr, page tvars_page)
High level function that initializes the graphics hardware. This function sets up a Toshiba TC6963C according to the variables in the tvars struct initialized by Config_Display. Init_Display dimensions the graphics and text arrays to the appropriate sizes in the display heap and enables the display hardware. Init_Display zeros the display_resource variable in the tvars struct.

Forth: Init_Menu (offset\ col\ row\ tvars_addr\ tvars_page\ menu xpfa --)
C: void Init_Menu(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr)
Draws and installs a menu at the given offsets. Offsets for the screen position, col and row, are applied to the relative locations of the objects contained in the menu. The resulting absolute screen locations are used to draw the objects to the screen. Init_Menu then calls Menu_Install. Each element of the menu has a relative button number associated with it which is added to the offset to get an absolute button number. The resulting absolute button number is the keymap array index in which the button object reference is stored by Menu_Install. This function is equivalent to passing the DRAW_ACTION flag to Do_Menu followed by a call to Menu_Install. When changing from one menu to another, you should call Uninit_Menu for the old menu before calling Init_Menu for the next menu. Afterwards, you must update the display since Uninit_Menu and Init_Menu do not automatically update the display. See Menu_Install, Uninit_Menu, and Do_Menu.

Forth: Init_Touch (tvars_addr\ tvars_page --)
C: void Init_Touch(GUI_VARS * tvars_addr, page tvars_page)
Initializes the touchscreen variables keymap_array, repeat_delay, and repeat_period in the tvars struct. It dimensions the keymap array for a 20 button touchscreen/keypad in the current heap. All the elements are then filled with 0x00. There must be a valid heap with enough room for the keymap array prior to calling this Init_Touch. Repeat_delay and repeat_period are initialized to 80 and 10 timeslice counts respectively. This assumes that the timeslicer period is set to its default value of 5 mS. If you change the timeslice period, may be necessary to adjust the repeat_delay and repeat_period variables to maintain desired operation. This function should be called as part of the start up initialization.

Forth: Menu_Install (offset\ col\ row\ tvars_addr\ tvars_page\ menu_xaddr --)
C: void Menu_Install(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr)
Copies each item from the menu object's array to the keymap array. Col, row, and offset determine the position of the menu on the display. Each element of the menu has a relative button number associated with it which is added to the offset to get an

absolute button number. The resulting absolute button number is the keymap array index in which the button object reference is stored. Offsets for the screen position location (col and row) are applied to the relative locations of the buttons contained in the menu. The resulting absolute screen locations are stored in the keymap array. Menu_Install does not draw the menu, but only places its buttons in the keymap so that the menu manager will be able to access them. This function is called by Init_Menu. See Wait_Then_Service_Touch, Menu_Remove, and Init_Menu.

Forth: Menu_Remove (offset\ tvars_addr\ tvars_page\ menu_addr\ menu_page --)
C: void Menu_Remove(uint offset, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr)

Removes a menu from the keymap array. Each element of the menu has a relative button number that is added to the offset to determine which button areas of the keymap array are occupied by the objects of the menu. Those areas are marked as unused by filling them with 0x00. Menu_Remove does not erase the menu from the screen. Uninit_Menu calls Menu_Remove. See **Uninit_Menu** and **Menu_Install** in the *Glossary of Functions*.

Forth: NEW_MENU: and BUILD_MENU
C: NEW_MENU and BUILD_MENU(arrayname, num_elements)

Declares a new menu array. There are syntactical differences in how this macro is used between C and forth. In C, NEW_MENU is actually a synonym for MENU_ENTRY, a structure type. Here is an example:

Forth:

```
NEW_MENU: mymenu_menu
\ Below, the Col and Row are the screen positions of the upper left
\ corners of the objects relative to the upper left corner of the menu.
\ The upper left corner of the menu on the screen is determined by the
\ values passed to Init_Menu.
\  Col Row   Action mask  Object xaddress     Object adding macro

0      38     DRAW_MASK    numshift_button     ADD_TOUCH_BUTTON
8      70     DRAW_MASK    numdec_button       ADD_TOUCH_BUTTON
24     102    DRAW_MASK    num0_button         ADD_TOUCH_BUTTON
3      32     DRAW_MASK    MY_LOGO             ADD_GRAPHIC
32     0      DRAW_MASK    mybutton1           ADD_TOUCH_BUTTON
BUILD_MENU
```

C:

```
NEW_MENU mymenu[5]=
{
// Below, the Col and Row are the screen positions of the upper left
// corners of the objects relative to the upper left corner of the menu.
// The upper left corner of the menu on the screen is determined by the
// values passed to Init_Menu.
// Object adding macro     (Col, Row, Action mask, Object xaddress)

ADD_TOUCH_BUTTON(  0,   38,  DRAW_MASK, numshift_button ),
ADD_TOUCH_BUTTON(  8,   70,  DRAW_MASK, numdec_button ),
ADD_TOUCH_BUTTON(  24, 102, DRAW_MASK, num0_button ),
ADD_GRAPHIC(           3,   32,  DRAW_MASK, MY_LOGO),
ADD_TOUCH_BUTTON(  32,  0, mybutton1)
};
BUILD_MENU( mymenu, 5);
```

The menus built above are identical. When programming in forth, NEW_MENU acts as a defining word that uses the stack to pass information to BUILD_MENU which instantiates the menu. This should be done at compile time, not inside a colon definition. Since C doesn't have the ability for two functions to communicate at compile time, it is important to restate the number of elements in the call to BUILD_MENU. The name of the final menu pointer in both cases is mymenu_menu. In C, a C style array called mymenu must first be created which is then used to create a forth array parameter field called mymenu_menu. BUILD_MENU automatically appends the suffix, _menu to the base name. This modified name is the name by which the menu should be referred in calls to menu related functions, not mymenu. In forth, the entire data structure is built at once and only has one name.

Forth: NORMBUTTON(flags\ draw_graphic_xaddr\ release_graphic_xaddr \press_graphic_xaddr \ handler\ label1\ label2\ label3\ label4 <name> --)

C: NORMBUTTON(uint flags, xaddr draw_graphic_xaddr, xaddr release_graphic_xaddr, xaddr press_graphic_xaddr, (void *) handler, char * label1, char * label2, char * label3, char * label4,<name>)

Creates a new button object. This macro integrates the creation and initialization of the BUTTON structure with some useful defaults. NORMBUTTON builds a button that uses all three graphics (draw, release, and press). By default, only DRAW_GRAPHIC_FLAG, RELEASE_GRAPHIC_FLAG, and PRESS_GRAPHIC_FLAG are set. Additional flags should be specified to further shape the button's behavior. These flags are ORed with the default flags. For the handler to be executed, you must specify PRESS_HANDLER_FLAG or RELEASE_HANDLER_FLAG. If both PRESS_HANDLER_FLAG and RELEASE_HANDLER_FLAG are specified, then the handler is executed twice, once when the button is pressed, and again when it is released. Other flags that might be useful are REPEAT_FLAG to make the button repeat or DRAW_TEXT_FLAG if the label text is to be printed in the button. The other parameters are stored in their respective fields in new button. The name given to the new button is specified by <name>. Here is an example:

Forth:

```
DRAW_TEXT_FLAG or                \ It has text
PRESS_HANDLER_FLAG or            \ It has a press handler
GRAPHICS_UPDATE_PRESS_FLAG or    \ Call Update_Graphics on press
GRAPHICS_UPDATE_RELEASE_FLAG or  \ Call Update_Graphics on release
LBLANK_PCX                       \ Graphic for DRAW_ACTION
LBLANK_PCX                       \ Graphic for RELEASE_ACTION
LBLACK_PCX                       \ Graphic for PRESS_ACTION
cfa.for myfunction               \ The code xaddress for press handler
" "                              \ Line 1 label
" Start"                         \ Line 2 label
" Pump"                          \ Line 3 label
" "                              \ Line 4 label
NORMBUTTON mybutton1             \ Instantiate the new button
```

C:

```
NORMBUTTON(
DRAW_TEXT_FLAG |                 // It has text
PRESS_HANDLER_FLAG |             // It has a press handler
GRAPHICS_UPDATE_PRESS_FLAG |     // Call Update_Graphics on press
GRAPHICS_UPDATE_RELEASE_FLAG,    // Call Update_Graphics on release
```

```
    LBLANK_PCX,                     // Graphic for DRAW_ACTION
    LBLANK_PCX,                     // Graphic for RELEASE_ACTION
    LBLACK_PCX,                     // Graphic for PRESS_ACTION
    myfunction,                     // The code address for press handler
    "",                             // Line 1 label
    "Start",                        // Line 2 label
    "Pump",                         // Line 3 label
    "",                             // Line 4 label
    mybutton1);                     // Instantiate the new button
```

Forth: Read_Touchscreen (tvars_addr \ tvars_page – n | 0 <= n <= 20)
C: int Read_Touchscreen(GUI_VARS * tvars_addr, page tvars_page)

Scans the touchscreen. If it is being pressed, returns the key number (1<=keynumber<=20); does not wait for a release. If nothing is being depressed, returns 0. Key 1 is in the upper left hand corner, key 2 is just below it, and key 20 is in the lower right hand corner.

Forth: Read_Raw_Coords (tvars_addr \ tvars_page – raw_coords)
C: long Read_Raw_Coords(GUI_VARS * tvars_addr, page tvars_page)

Reads raw touchscreen values. The raw values are used by Calibrate_Touchscreen to calibrate the touchscreen. See Calibrate_Touchscreen.

Forth: Service_Touch (button number \ tvars_addr \ tvars_page --)
C: void Service_Touch(int button, GUI_VARS * tvars_addr, page tvars_page)

This routine is very similar to Wait_Then_Service_Touch. Instead of polling the hardware and then reacting as Wait_Then_Service_Touch does, this function accepts a button number as collected by the calling environment using one of the built-in kernel functions. Service_Touch then processes the button exactly as Wait_Then_Service_Touch does, and if the button is being held down, then Service_Touch blocks until it is released. This routine effectively serves as a non-blocking version of Wait_Then_Service_Touch. See Wait_Then_Service_Touch. The following examples show how to implement Wait_Then_Service_Touch yourself using Service_Touch and the builtin kernel driver for the keypad:

C:

```
void My_Wait_Then_Service_Touch ( GUI_VARS * tvars_addr, page tvars_page )
{
  int this_press=0;      // Init to default
  while (this_press==0)  // Keep looping until a button is pressed
  {
    this_press = Read_Touchscreen();
  }
  Service_Touch( this_press, TVARS );  // we have a button press!  Act on it
}
```

Forth:

```
: my_Wait_Then_Service_Touch ( tvars -- )
  locals{ x&tvars }
  begin
    Read_Touchscreen  \ ( [ button number \ true ] or [ false ] -- )
    if                \ ( tvars \ button number -- )
      x&tvars Service_Touch  \ Process the button number received
      true            \ cause an exit
    else
      false           \ cause the loop to continue
    endif
  until               \ loop if nothing yet
```

;

Forth: Set_Cursor_State (isvisible\ isflashing --)
C: void Set_Cursor_State(boolean isvisible, boolean isflashing)
Calls Set_Display_Mode to sets the state of the cursor using the 2 flags. Isvisible is true if the cursor is visible and false if not. Isflashing is true for flashing and false for non-flashing. See PutCursor (forth: put.cursor) in the main glossary.

Forth: Set_Display_Mode (mode --)
C: void Set_Display_Mode(uint mode)
Sets the mode byte of the display controller. This word uses the lower 4 bits of mode to determine the operating mode. Details about the meaning of the mode flag can be found in the datasheet for the TC6963 controller, but it is handled for you by Set_Cursor_State and Set_Display_State which call Set_Display_Mode.

Forth: Set_Display_State (graphics\ text --)
C: void Set_Display_State(boolean graphics, boolean text)
Calls Set_Display_Mode to enable or disable graphics or text according to the 2 flags. Graphics is true to indicate that the graphics layer is enabled and text is true to indicate that the text layer is enabled. Init_Display sets this automatically based on the display configuration specified by Config_Display. Either text or graphics may be written to the display even when that layer has been disabled with this function. The layer may then be re-enabled to make visible the data currently stored in the display. This may be useful for blanking the screen during an update. Most applications don't need this ability.

Forth: Set_Gr_Area (columns --)
C: void Set_Gr_Area(uint columns)
Sets the width for graphics in the TC6963 display controller. The graphics area value should be equal to the number of graphics columns. This is not the same as the number of pixels of display width, but the number of bytes required to represent a line of graphics. The 240x128 display is configured for 6 bits per byte meaning that the number of columns is 240/6 or 40, the same as text. Init_Display sets this automatically.

Forth: Set_Gr_Home_Addr (address --)
C: void Set_Gr_Home_Addr(addr address)
Sets the home address for graphics in the TC6963 display controller. Only the rarest of circumstances require altering the display's internal memory configuration. The address specified will become the starting address inside the display module for the graphics data. It is set automatically by Init_Display.

Forth: Set_Pixel (x\y --)
C: void Set_Pixel(uint x, uint y)
Sets a pixel directly to the LCD display bypassing the graphics array. Since it writes directly to the screen, and not to the graphics array, calling Update_Graphics will overwrite anything set on the screen by Set_Pixel. See Clear_Pixel.

Forth: Set_Text_Area (columns --)
C: void Set_Text_Area(uint columns)
Sets the width for text in the TC6963 display controller. The text area value should be equal to the number of character columns. Characters are 6 pixels wide meaning that there are 240/6 or 40 text columns on the display. Init_Display sets this automatically.

Forth: Set_Text_Home_Addr (address --)
C: void Set_Text_Home_Addr(addr address)
Sets the home address for text in the TC6963 display controller. Only the rarest of circumstances would require altering the display's internal memory configuration. The address specified will become the starting address inside the display module for the text data. It is set automatically by Init_Display.

Forth: Set_Text_Mode (modebyte --)
C: void Set_Text_Mode(uchar modebyte)
Sets the text attribute bits. Only the lower 4 bits used, and the other bits are ignored. Bit 3 is 0 for character generator ROM mode and 1 for character generator RAM mode. Unless you are using custom fonts uploaded to the display, this bit should be 0. If it is set to 1, then the cg offset pointer, a register in the TC6963 display controller chip, must be set to point to the base address of the character table in the display's RAM. An example of how to do this may be found in the fonts directory of the distribution of the GUI Toolkit package. Bits 2, 1, and 0 determine the display mode for text.

Constant	Bit2	Bit1	Bit0	Description
OR_TEXT	0	0	0	Text is ORed with graphics (default)
EXOR_TEXT	0	0	1	Text is EXORed with graphics
AND_TEXT	0	1	1	Text is ANDed with graphics
	1	0	0	Text is in special attribute mode. This specialized mode is not usable with graphics mode and is not discussed here. See TC6963 Datasheet for more info.

Forth: Std_Display (tvars_addr\ tvars_page --)
C: void Std_Display(GUI_VARS * tvars_addr, page tvars_page)
Sets the display configuration information in the tvars struct to default values for the 240x128 display used on the QScreen controller. When using such a display, simply calling this function prior to calling Init_Display will eliminate the need to use Config_Display which can be unwieldy. This function calls Config_Display with the following parameters. Don't forget to call Init_Display after calling Std_Display.

graphic_cols, graphic_rows -- 40, 128
graphics_start -- 0x0280
background_fill -- 0
text_cols, text_rows -- 40, 16
text_start -- 0x0000
heap_bottom -- 0x0F47FF
heap_top -- 0x0F3000

See **Init_Display** and **Config_Display** in the *Glossary of Functions*.

Forth: Uninit_Menu (offset\ col\ row\ tvars_addr\ tvars_page\ menu xpfa --)
C: void Uninit_Menu(uint offset, uint col, uint row, GUI_VARS * tvars_addr, page tvars_page, MENU * menu_xaddr)
Erases and uninstalls the menu at the given offsets. Offsets for the screen position, col and row, are applied to the relative locations of the objects contained in the menu. The resulting absolute screen locations are used to erase the objects from the screen. Uninit_Menu then calls Menu_Remove. Each element of the menu has a relative button number associated with it which is added to the offset to get an absolute button number. The resulting absolute button number is the keymap array index in which the button object reference is deleted by Menu_Remove. This function is equivalent to calling Do_Menu with the ERASE_ACTION flag followed by a call to Menu_Remove. When changing from one menu to another, you should call Uninit_Menu for the old menu before calling Init_Menu for the next menu. Afterwards, you must update the display since Uninit_Menu and Init_Menu do not automatically update the display. See Menu_Remove, Init_Menu, and Do_Menu

Forth: Update_Graphics (tvars_addr\ tvars_page --)
C: void Update_Graphics (GUI_VARS * tvars_addr, page tvars_page)
Calls Update_Here_With to send the entire contents of the graphics array to the LCD display. Call this function after modifying the graphics array to update the display. The contents of the graphics array are transferred to the memory address inside the display specified by Gr_Home_Addr in the tvars struct. Any graphics that were drawn directly to the LCD bypassing the graphics array will be overwritten when Update_Graphics is called.

Forth: Update_Here_With (address\ graphics_resource_addr\ graphics_resource_page\ garray_xaddr --)
C: void Update_Here_With(addr address, addr * graphics_resource_addr, page graphics_resource_page, FORTH_ARRAY * garray_xaddr)
Directly copies the contents of the 2 dimensional array pointed to by garray_xaddr to the display starting at address in the display's memory. This function honors the resource variable pointed to by graphics_resource_addr and graphics_resource_page. The display resource must be available or this function will hold up execution until it can take control of the display to perform the update. Update_Here_With is a low level function that shouldn't be needed in most circumstances. It is called by Update_Text and Update_Graphics. Update_Here_With is useful for loading special areas of the display memory with data such as custom fonts. See Update_Text and Update_Graphics.

Forth: Update_Text (tvars_addr\ tvars_page --)
C: void Update_Text(GUI_VARS * tvars_addr, page tvars_page)
Calls Update_Here_With to send the entire contents of the text array to the LCD display. Call this function after modifying the text array to update the display. The contents of the text array are transferred to the memory address inside the display specified by Text_Home_Addr in the tvars struct.

Forth: Update_Text_And_Graphics (tvars_addr\ tvars_page --)
C: void Update_Text_And_Graphics(GUI_VARS * tvars_addr, page tvars_page)
Sends the contents of the text and graphics arrays to the LCD display. This function is the equivalent of calling Update_Text and Update_Graphics. See Update_Text and Update_Graphics

Forth: Wait_For_Press (tvars_addr\ tvars_page --)
C: void Wait_For_Press(GUI_VARS * tvars_addr, page tvars_page)
Waits until a press is detected on the touchscreen before returning. See Wait_For_Release and Read_Touchscreen.

Forth: Wait_For_Release (tvars_addr\ tvars_page --)
C: void Wait_For_Release(GUI_VARS * tvars_addr, page tvars_page)
Waits until a release of the touchscreen before returning. See Wait_For_Press and Read_Touchscreen.

Forth: Wait_Then_Service_Touch (tvars_addr\ tvars_page --)
C: void Wait_Then_Service_Touch(GUI_VARS * tvars_addr, page tvars_page)
This routine serves as a runtime menu manager for monitoring the user input hardware (touchscreen or keypad). It waits for a keypad or touchscreen press. When a button is pressed, held, or released the touchscreen/keypad hardware driver returns a button number which is used as an index to the keymap array.
Wait_Then_Service_Touch then examines the indexed element of the keymap array and invokes the PRESS_ACTION, REPEAT_ACTION, or RELEASE_ACTION to the object whose xaddress is stored in the indexed keymap array element.

Wait_Then_Service_Touch does not loop. After one press/release cycle, it exits. If the button has the REPEAT_FLAG set, then Wait_Then_Service_Touch invokes the REPEAT_ACTION to the button repeatedly according to the repeat times, represented in timeslicer counts, stored in the repeat_period and repeat_delay variables in the tvars struct. The timeslicer must be running for buttons to repeat. Wait_Then_Service_Touch is usually used inside a loop that iterates for each press/release cycle. Wait_Then_Service_Touch calls Pause while awaiting a button press. Any user handlers associated with the buttons will run under the same task as Wait_Then_Service_Touch since Wait_Then_Service_Touch executes the code.