APPLICATION NOTE

AN411

Color Touchscreen Application Kit

The Color Touchscreen Application Kit contains the hardware and software for creating an easy-to-use graphical interface for device monitoring and control. A graphics library, some predefined bitmaps and a variety of sample programs provide the tools you'll need to program the Rabbit-based controller to display stationary and "animated" graphics on the Reach Technology SLCD Graphics Touch Terminal (GTT).

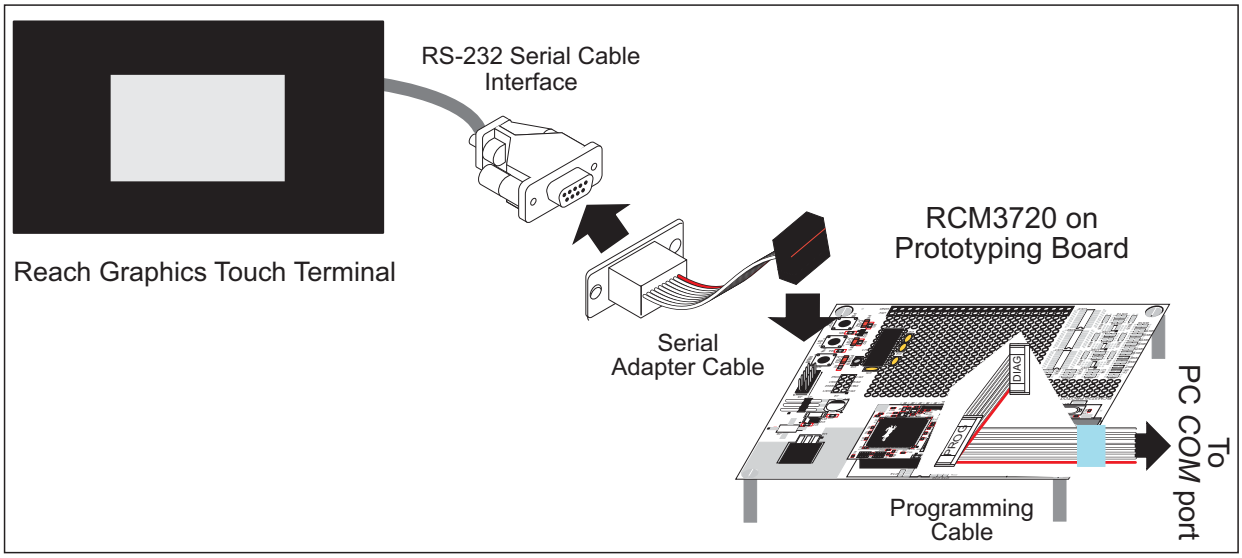
Hardware Components

The Color Touchscreen Application Kit has the following hardware:

- RCM3720 RabbitCore module
- RCM3720 Prototyping Board with RS-232 circuits installed.
- Reach Technology GTT

The hardware connection between the RCM3720 and the GTT are shown in Figure 1, and are explained in detail in the *Color Touchscreen Application Kit Getting Started* instructions (located on the supplemental Dynamic C CD).

Figure 1. Color Touchscreen Hardware Interface



Any applications developed using the Color Touchscreen Application Kit can be easily ported to any Rabbit-based single-board computer or RabbitCore module with an RS-232 interface. For a complete description of the RCM3720 and the prototyping board included in this kit, refer to the *RCM3700 User's Manual* (located on the Dynamic C CD).

Software and Documentation

The Color Touchscreen Application Kit comes with three CD-ROMs:

1. Dynamic C CD contains:

    - Dynamic C installation
    - Dynamic C documentation and RCM3720 documentation

2. Supplemental Dynamic C CD contains:

    - Application Kit sample programs
    - *Color Touchscreen Application Kit Getting Started* instructions

3. Reach Technology CD contains:

    - `BMPload.exe`, a Windows program for downloading images and macros to the GTT
    - `SGTT.pdf`, *SLCD Graphics Touch Terminal Technical Manual*: the user's manual for the Reach GTT
    - `AN100.pdf`, Reach Technology document describing a full-featured application that uses the low-level Reach commands instead of the API functions provided with the Color Touchscreen Application Kit.

Definitions

This section clarifies some terms.

bitmap

A graphical image created using a program such as Windows Paint. The bitmaps are downloaded to the Reach GTT via the Windows program `BMPload.exe`. This makes them available for display or for use as a button object.

button

A defined area of the screen that is touch sensitive (see "touch area") and has two bitmaps associated with it, one for when the button is not pressed and the other for when it is pressed.

latched button

Type of button that operates in a push on/ push off mode.

momentary button

Type of button that is activated for as long as the operator is pressing it.

typematic

Operation associated with a momentary button. It is a feature that causes the button notification to be repeated for as long as the button is pressed.

touch area

An area of the screen defined by x and y coordinates that has a user-assigned number associated with it. When the area is touched, a message is sent from the Reach controller to the Rabbit. A total of 128 touch areas are allowed.

Sample Programs

The software included with the supplemental Dynamic C CD contains the bitmaps and macros that need to be loaded to the Reach Technology display before the sample programs listed below can be run. A summary of the loading process is described in the *Color Touchscreen Application Kit Getting Started* instructions. For complete details on the loading process refer to Appendix A of the *SLCD Graphics Touch Terminal Technical Manual*.

When you installed the supplemental Dynamic C CD, you also installed some sample programs that illustrate the use of the RCM3720 with the Reach Technology GTT. These sample programs can be found in the Dynamic C `Samples\ColorTouchscreen` folder.

> **TIP:** Use the bitmaps, macros, and sample programs from the Dynamic C supplemental CD. While these are based on the bitmaps and the sample program included with the Reach Technology CD, the sample programs on the Dynamic C supplemental CD contain all the latest enhancements.

Each sample program has comments that describe the purpose and function of the program. Follow the instructions in the comments at the beginning of the sample program. Note that the RCM3720 must be installed on the Prototyping Board when using these sample programs, and it must be connected to the GTT as shown in Figure 1 after the bitmaps and macros have been loaded.

To run a sample program, open it with the File menu, then compile and run it by pressing F9. The following sample programs are available.

`LCD_BAR_CHART1.C, LCD_BAR_CHART2.C, LCD_BAR_CHART3.C`

Various demos of bar charts.

`LCD_BUTTON1.C, LCD_BUTTON2.C, LCD_BUTTON3.C`

Various demos of input to the GTT using buttons.

`LCD_PONG1.C, LCD_PONG2.C`

Colorful take on the familiar pong demo.

`LCD_TEXT1.C`

Demo of text moving across the screen.

`SLCD_DEMO.C`

A comprehensive demo that exercises many of the kit's API functions.

Source Code Walk-Through

In this section the sample program LCD_BUTTON1.C is used to show how easy it is to create and display a button that will notify your application when it has been pressed. We will only examine some of the code; to view the source code for the entire program you must open it in Dynamic C or a text editor.

One of the first things that must be done in any application written for the Color Touchscreen Application Kit is initializing the serial port communication channel between the Rabbit-based controller and the Reach GTT. The graphics library provides some macros that make the serial port initialization very easy. All you have to do is #define one of the serial port macros. For example, if Serial Port D is used to connect the Reach GTT to the Rabbit, the application must include the statement:

```
#define LCD_USE_PORTD                    // tells library which serial port to use
```

Every button must be identified by an integer in the range of 1 to 127.

```
#define BUTTON1 1                        // Create a button ID for button function
```

Buttons have two states: on (pressed) and off (unpressed); therefore, each button must have two associated images. The images are identified by their position in demo.lst, a file that defines which BMPs are copied to the Reach GTT. If you open \samples\ColorTouchscreen\BMP_Macro\demo.lst, you will see that the .bmp files we want to display are the 30th and 31st files in the list.

```
#define BMP_big_button 30
#define BMP_big_button_dn 31
```

All sample programs or user-defined applications that call API functions from this kit's graphics library must include the statement:

```
#use REACH.LIB
```

The sample program includes some board setup calls to allow the sample program to be used with all Rabbit-based single-board computers and RabbitCore modules. When an RCM3600 or RCM3700 series RabbitCore module is used with the RCM3700 Prototyping Board, PE5 must be toggled low to enable the RS-232 chip since the RCM3700 Prototyping Board Serial Ports C and D are also used with the IrDA transceiver on the RCM3700 Prototyping Board.

```
   #if BOARD_FAMILY == RCM36
   BitWrPortI(PEDR, &PEDRShadow, 0, 5); // set low to enable rs232 device
   #endif
```

The BL2600 series of single-board computers has its configureable DIO0–DIO3 set as inputs and DIO4–DIO7 set as outputs for use with the Demonstration Board.

```
   #if BOARD_FAMILY == BL26
   digOutConfig ( 0x00F0 );        // DIO0-3 = Input,  DIO4-7 = Output
   #endif
```

The main loop of the program is straight-forward:

```
main() {
   int pressed_button;             // variable to store which button has been pressed

   brdInit();                      // initialize board for this demo
   setup_lcd();                    // Connect with LCD, setup colors, and clear screen
```

```
   while (1) {
      lcd_ClearScreen();          // Clear screen
      drawButtons();

      // wait for button press & release
      while( (pressed_button = lcd_GetTouch(100)) == -1 );
      if (pressed_button == BUTTON1) {
         lcd_ClearScreen();
         lcd_DispText("You got me!", 100, 100, MODE_NORMAL);
         fnMsDelay(1000);              // delay long enough to read displayed text
         pressed_button = 0;           // reset pressed button value
      }
   }
}
```

The function `brdInit()` is familiar to most Dynamic C users. It is a board-specific function that initializes I/O and system ports. The next function call, `setup_lcd()` is a local function that was defined prior to `main()`. A call to `lcd_Connect()` is made by `setup_lcd()` in order to establish a serial communication channel between the RCM3720 and the GTT. Once the communication channel is operational, details like font choice and screen colors are requested.

With the initialization procedures completed, the sample program enters an endless while loop. The GTT screen is cleared (`lcd_ClearScreen()`) and a call is made to the local function `drawButtons()`.

```
void drawButtons()
{
   // define and draw a button
   lcd_ButtonDef(BUTTON1,                  // BUTTON NUMBER 1
                 BTN_MOM,                  // momentary button
                 BTN_TLXY, 120, 80,        // top left x-y coordinates
                 BTN_TYPE, BUTTON_PRESS,   // notify on press
                 BTN_TEXT, "Hit me!",      // text in button
                 BTN_TXTOFFSET, 5, 20,     // text x-y coordinates
                 BTN_BMP, BMP_big_button, BMP_big_button_dn,
                 BTN_END);
}
```

The API function `lcd_ButtonDef()` allows you to associate graphic images (`BMP_big_button` and `BMP_big_button_dn`) with a button number (`BUTTON1`). You can locate the button anywhere on the screen by specifying the top left *x* and *y* coordinates. You can also superimpose text on the button. See the function description for `lcd_ButtonDef()` for other button parameters that you can customize.

After the button is displayed on the Reach GTT, the program waits for it to be pressed:

```
   while( (pressed_button = lcd_GetTouch(100)) == -1 );
```

The program will stay in the while loop until it receives notification from the GTT that a screen press has taken place. It then checks to make sure the expected button was pressed before responding to it. The response is to remove the button graphic before displaying some text (`lcd_DispText()`). A delay is added (`fnMsDelay()`) to give the user time to read the text. Then the screen is cleared, the button is redrawn, and the program again waits for a button press notification.

Dynamic C Functions

The functions described in this section are for use with the Color Touchscreen Application Kit. The source code is in \LIB\DISPLAYS\ColorTouchscreen\REACH.LIB.

All the function calls are nonreentrant and blocking. Each function call is associated with a command for the Reach GTT. These commands are documented in the *Graphics Touch Terminal Technical Manual*.

Several of the API functions in REACH.LIB take a variable number of parameters. These optional parameters come in groups, with the first parameter in the group being one of a documented set of identifiers, and subsequent parameters in the group being the value specific to that identifier. The list of parameter groups must be terminated using a parameter that signifies the end of the parameter list. Here is an example:

```
lcd_Chart(0,                          //  chart id number
        CHART_TLXY, 50, 50,           //  left/top corner
        CHART_BRXY, 70, 200,          //  right/bottom corner
        CHART_BKG, lcd_LTGRAY_D,      //  background color
        CHART_PEN 2, lcd_DKRED_D,     //  pen width and color
        CHART_END);                   //  end of parameter list
```

---

fnMsDelay

```
void fnMsDelay( unsigned int tdly );
```

DESCRIPTION

Creates a delay of the specified number of milliseconds.

PARAMETER

tdly Number of milliseconds to wait

RETURN VALUE

None.

lcd_Backlight

```
int lcd_Backlight( int brightness );
```

DESCRIPTION

Modifies the backlight brightness.

PARAMETER

brightness The backlight brightness (0–5, where 5 is the brightest).

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): illegal brightness value
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Bar

```
int lcd_Bar( int bar_nbr, ..., BAR_END );
```

DESCRIPTION

This function is used in two ways—it can be used to define a bar graph, with default values for each of the optional parameters; and it can be used to update the defined bar graph with specified values using the BAR_VALUE parameter.

PARAMETERS

bar_nbr the number of the bar graph (0–9)

The remaining parameters are optional and may be used in any order (unless otherwise specified). The parameter name and value must be entered as follows:

```
<parameter name>, <parameter value>, [<parameter value2>,]
```

BAR_END must be the final parameter

BAR_VALUE (float) This macro is a special case; if it is the first "optional" parameter, no other parameters are evaluated, and the value is displayed on the bar chart. The execution of this option *must* be preceded by a call to this function without BAR_VALUE since the first parameter BAR_END is optional for this special case.

After normalization using BAR_MAX_VALUE (top/right) and BAR_MIN_VALUE (bottom/left) the float value is converted to an integer for transmission to the display.

The chart will not be displayed on the screen until a BAR_VALUE is specified.

NOTE: Defining any of the following parameters will cause several of the parameters not being defined in the statement to take on their default values (unless otherwise indicated).

BAR_TLXY (int) two values: top left *x* and *y* coordinates in pixels, this is the coordinate closest to (0,0), the top left corner of the GTT. The default values start at (9,10), and will be updated based on BAR_SEP. This parameter must precede BAR_BRXY or BAR_SIZE, whichever is used.

BAR_BRXY (int) two values: bottom right x and y coordinates in pixels. If these values are specified, they will be used to calculate the BAR_SIZE values.

BAR_SIZE (int) two values: x and y pixel sizes. The default values are (50,100). Any time this parameter is specified, its value becomes the default.

BAR_SEP (int) two values: x and y separation in pixels relative to BAR_TLXY. This is only used if you want multiple bars evenly spaced and do not want to specify the BAR_TLXY for each. The default values are (64,110). Any time this parameter is specified, its values become the default.

BAR_MAX_VALUE (float) value at “top” of bar; default = 100.0.

BAR_MIN_VALUE (float) value at "bottom" of bar; default = 0.0.

BAR_SEG (int) percentage of full scale for the start of upper segment and/or the middle segment; maximum value = 100, default is one green segment from 0% to 100%

The bar graph may have as many as three segments. If no segments are specified, there will be one segment.

One BAR_SEG parameter:

Top/right segment: Green from specified value to 100%
Bottom/left segment: Yellow from 0% to specified value

Two BAR_SEG parameters:

Top/right segment: Green from first specified value to 100%
Middle segment: Yellow from 2nd specified value to first specified value
Bottom/left segment: Red from 0% to second specified value

For proper operation the higher percentage value must be listed first. BAR_SEG must *not* be specified for the top segment.

BAR_SEGCOLOR (int) the color associated with the last defined BAR_SEG. See BAR_SEG for the default colors. Any time this parameter is specified, its value becomes the default.

See the Reach Technology manual for Set Color (Detailed) for more information.

BAR_BORDER (int) width, in pixels, of the border; default = 1. Any time this parameter is specified, its value becomes the default.

Bar Orientation Use only one of the following two macros with no parameter.

BAR_VERT vertical orientation (default), the minimum value is at the bottom

BAR_HORIZ horizontal orientation, the minimum value is on the left

If no orientation is specified, the function will determine the orientation automatically based on the following formula.

```
if (BRx - TLx)/(BRy - TLy) ≤ 1.0
   orientation = BAR_VERT
```

BAR_BKG (int) background color; default = gray (0x888). Any time this parameter is specified, its value becomes the default.

See the section titled “Set Color (Detailed)” in the *SLCD Graphics Touch Terminal Technical Manual* (sgtt.pdf) for more information.

lcd_Bar (continued)

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): illegal parameter
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

EXAMPLE

```
lcd_Bar( 0,
    BAR_TLXY, 50, 50,
    BAR_SIZE, 50, 100,
    BAR_BKG, lcd_LTGRAY_D,
    BAR_SEGCOLOR, lcd_RED_D,          // force top segment to red
    BAR_SEG, 70,                      // middle segment ends at 70%
    BAR_SEGCOLOR, lcd_YELLOW_D,       // middle segment to yellow
    BAR_SEG, 30,                      // bottom segment ends at 30%
    BAR_SEGCOLOR, lcd_GREEN_D,        // bottom segment to green
    BAR_END );
lcd_Bar( 0, BAR_VALUE, 50.5 );        // display a value on bar chart
```

lcd_BeepVolume

```
int lcd_BeepVolume( int Volume );
```

DESCRIPTION

Sets the beep volume and saves the setting in global variable `lcd_beepvolume`.

PARAMETER

`Volume` The beep volume. Valid values are in the range 0–255, with 255 being the maximum volume.

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-(parameter number)`: illegal volume
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_BFcolorsB

```
int lcd_BFcolorsB( int ForeColor, int Backcolor );
```

DESCRIPTION

Sets the default basic background and foreground colors.

The colors have been predefined:

:

lcd_BLACK	lcd_WHITE	lcd_BLUE	lcd_GREEN
lcd_CYAN	lcd_RED	lcd_MAGENTA	lcd_BROWN
lcd_DGREY	lcd_GREY	lcd_LGREY	lcd_LBLUE
lcd_LGREEN	lcd_LCYAN	lcd_LRED	lcd_LMAGENTA
lcd_YELLOW			

PARAMETERS

ForeColor foreground color

BackColor background color

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal color
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_BFcolorsD

```
int lcd_BFcolorsD( int ForeColor, int Backcolor );
```

DESCRIPTION

Sets the detailed background and foreground colors as explained in the section titled “Set Color (Detailed)” in the *SLCD Graphics Touch Terminal Technical Manual*.

Some of the available colors have been predefined:

lcd_RED_D	lcd_DKRED_D	lcd_GREEN_D	lcd_DKGREEN_D
lcd_BLUE_D	lcd_DKBLUE_D	lcd_YELLOW_D	lcd_GRAY_D
lcd_LTGREY_D	lcd_MDGRAY_D	lcd_VLTGRAY_D	

PARAMETERS

ForeColor foreground color

BackColor background color

RETURN VALUE

lcd_SUCCESS: full message received
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_ButtonDef

```
int lcd_ButtonDef( int ButtonID, ... BTN_END );
```

DESCRIPTION

Defines a button. The size of the button is determined by its associated bit map. The GTT sends a notification when a screen touch is detected; the notification is the string referred to in the function description for `lcd_GetTouch()`.

notification on button press = x<n><return>
notification on button release = r<n><return>
<n> = `ButtonID`

PARAMETERS

`ButtonID` Identifies a button. Must be in the range of 1 to 127.

The remaining parameters are optional; if they are not passed to the function default values will be used. The parameters may be used in any order (unless otherwise specified). With the exception of the operating mode parameters, the parameter name and value must be entered as follows:

```
<parameter name>, <parameter value>, [<parameter valueN>,]
```

`BTN_END` must be the final parameter when not using `BAR_VALUE`

The following two parameters define the button's operating mode. They are mutually exclusive. The operating mode should immediately follow `ButtonID`.

`BTN_MOM` This defines the button operation as "momentary," and is the default. Any time this value is specified, it becomes the default.

`BTN_LAT` This defines the button operation as "latched." Any time this value is specified, it becomes the default.

The coordinates can be forced using the following three parameters. Note that the (0,0) location is the top left corner of the LCD.

`BTN_TLXY` (`int`) two values: top left *x* and *y* coordinates. This is the coordinate closest to (0,0). If no values are entered, the button will be placed at an *x* offset `BTN_SEP` from the previous *x* value. If this is greater than `BTN_MARGINS` (second value), the *x* will be reset to `BTN_MARGINS` (first value) and the *y* value will be increased by `BTN_SEP`. Both values are initialized to 10.

`BTN_SEP` (`int`) two values: allows you to specify the *x* and *y* separation values for placing multiple buttons without having to specify `BTN_TLXY` for each. The separation is defined as the distance between the top left corners of the bitmaps. Any time this parameter is specified, its value becomes the default. Both values are initialized to 80. (Note that the buttons will "draw" in the *x* direction, first then the *y* direction.)

BTN_MARGINS (int) two values: the left and right margins for automatic positioning of multiple buttons. See BTN_TLXY. Any time this parameter is specified, its value becomes the default. The values are initialized to (10, 310); MIN value = 0, MAX value = 339.

BTN_TYPE (int) one value: button type; its default value is BUTTON_RELEASE. Any time BTN_TYPE is specified, its value becomes the default. The parameter value can be one of the macros listed under "Momentary" or "Latched."

Momentary:

- BUTTON_PRESS: notify on press
- BUTTON_TYPEMATIC: notify on press - typematic function
- BUTTON_RELEASE: notify on release (default)
- BUTTON_PR: notify on press and release

Latched:

- BUTTON_LAT: latching: display appropriate bitmap (see BTN_BMP) for off and on states.
- BUTTON_LAT_0: initialize to state 0
- BUTTON_LAT_1: initialize to state 1

BTN_TEXT BTN_MOM: (char*) address of the button text.

BTN_LAT: (char*) two values for the addresses of the button text. The first value is button off; the second value is button on.

The current limit for the button text string is 19 characters total.

BTN_TXTOFFSET BTN_MOM: (int) two values: *x* and *y* offset values within the button for text.

BTN_LAT: (int) four values: *x* and *y* offset values within the button for the text. The first two values are for button off text; the second two values are for button on text.

Any time this parameter is specified, its values become the default. The values are initialized to 5.

BTN_BMP (int) two values: bit map numbers, the first one is for the off state and the second one is for the on state. Any time this parameter is specified, its values becomes the default. The values are initialized to 22 and 23.

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal button
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_Calibrate

```
int lcd_Calibrate( void );
```

DESCRIPTION

Runs a calibration procedure whereby the user is asked to touch points on the screen to calibrate it. The calibration values are stored in non-volatile memory and restored on power-on.

This command will not cause the program to block during calibration. It is the responsibility of the programmer or user to ensure that the calibration is complete before using the LCD/touch-screen. The following code is one way to do this.

```
while (lcd_Origin(0,0) != lcd_SUCCESS)
   fnMsDelay(1000);                          // wait for done calibrate
```

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Chart

```
int lcd_Chart( int chart_nbr, ..., CHART_END );
```

DESCRIPTION

This function is used in two ways—it can be used to define a stripchart that will then be displayed on the GTT; and it can be used to update the defined stripchart with specified values using the CHART_VALUE parameter. There are default values for each of the optional parameters.

PARAMETERS

chart_nbr number of the stripchart (0–4)

The remaining parameters are optional and may be used in any order (unless otherwise specified). The parameter name and value must be entered as follows:

```
<parameter name>, <parameter value>, [<parameter value2>,]
```

CHART_END must be the final parameter when not using CHART_VALUE

CHART_VALUE (float) displays chart values. This is a special case; if it is the first optional parameter, no other parameters are evaluated and the values are displayed on the stripchart. There *must* be a value for each of the defined pens. Any parameters following the values will be ignored. The execution of this option *must* be preceded by a call to this function without CHART_VALUE. After normalization using CHART_MAX_VALUE and CHART_MIN_VALUE, the float value is converted to an integer for transmission to the display.

NOTE: Defining any of the following parameters will cause all the parameters not being defined in the statement to take on their default values.

Default coordinates are calculated based on the chart_nbr assuming one column of five evenly spaced stripcharts. The coordinates can be forced using the following two parameters. Keep in mind that the (0,0) location is the top left corner of the LCD.

CHART_TLXY (int) two values: top left x and y coordinates, this is the coordinate closest to (0,0). The default values will place up to five stripcharts. The starting (x,y) for a stripchart can be calculated as follows:

```
x = 20, y = (stripchart number * 45) + 10
```

CHART_BRXY (int) two values: bottom right *x* and *y* coordinates. The default values will create a stripchart 180 pixels wide and 40 pixels high.

CHART_MAX_VALUE (float) value at “top” of bar; default = 100.0.

CHART_MIN_VALUE (float) value at “bottom” of bar; default = 0.0.

CHART_PEN two values:

(int) pen width in pixels: 1 or 2; default is 1
(int) the color for the pen, default is dark red. Several of the available colors have been defined in this library (see lcd_BFcolorsD()). See the section titled "Set Color (Detailed)" in the *SLCD Graphics Touch Terminal Technical Manual* for those colors that are not defined in reach.lib.

The stripchart may have as many as three pens. The pen number is identified by the order it appears in the parameter list; in the example below pen 1 and pen 2 are defined. If none are specified, there will be one pen. If more than one pen is desired, you *must* define values for pen 1.

CHART_DATA_WIDTH (int) number of horizontal pixels per data point; default = 4.

CHART_BKG (int) background color, default is dark blue. See the section titled "Set Color (Detailed)" in the *SLCD Graphics Touch Terminal Technical Manual* for more information.

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal parameter
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

EXAMPLE

```
lcd_Chart (0,
    CHART_TLXY, 50, 50,
    CHART_BRXY, 70, 200,
    CHART_BKG, lcd_LTGRAY_D,
    CHART_PEN, 2, lcd_DKRED_D,
    CHART_PEN, 2, lcd_DKBLUE_D,
    CHART_END );
```

lcd_Circle

```
int lcd_Circle( int x, int y, int radius, int filled );
```

DESCRIPTION

Draws a circle using the current foreground color.

PARAMETERS

x	center x coordinate in pixels
y	center y coordinate in pixels
radius	radius in pixels
filled	<0: not filled, line width is 1 pixel 1: filled with current foreground color

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_ClearScreen

```
int lcd_ClearScreen( void );
```

DESCRIPTION

Clears the screen to the current background color.

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Connect

```
int lcd_Connect( void );
```

DESCRIPTION

Opens the serial port and initializes communication with the display. The default baud rate is 115200 bps. Before executing this function, you must first define which serial port to use by defining the macro `LCD_USE_PORT<serial port designator A..F>`; for example,

```
#define LCD_USE_PORTB
```

There is a fixed 200 ms delay to allow the GTT to initialize.

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Cursor

```
int lcd_Cursor( int xval, int yval );
```

DESCRIPTION

Sets the cursor location for subsequent text commands; coordinates are relative to origin.

PARAMETERS

xval x coordinate (0–319)

yval y coordinate (0–239)

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): illegal coordinate
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_DefTouchArea

```
int lcd_DefTouchArea( int IDnum, ... TA_END );
```

DESCRIPTION

Sets a touch area on the GTT or defines the touchmatic area. When the touch area is pressed, the GTT will send an "x" response followed by the touch area `IDnum`.

PARAMETERS

`IDnum` Touch area ID number to set (must be between 128 and 255).

The remaining parameters are optional and may be used in any order (unless otherwise specified). The parameter name and value must be entered as follows:

`<parameter name>, <parameter value>,`

`TA_END` must be the final parameter

The coordinates can be forced using the following two parameters. Keep in mind that the (0,0) location is the top left corner of the LCD. If both values are 0, the area will be placed at the location specified by the last `lcd_Origin` statement.

`TA_TLXY` (`int`) two values: top left *x* and *y* coordinates. This is the coordinate closest to (0,0). Note that if both values are 0, the area will be placed at the location specified by the last `lcd_Origin` statement. If no values are entered, the area will be placed at an *x* offset `TA_SEP` from the previous *x* value. If this is >300, *x* will be reset to 0, and the *y* value will be increased by `TA_SEP`. Both values are initialized to 10.

`TA_SIZE` (`int`) two values: the *x* and *y* size of the area. Any time this parameter is specified, its value becomes the default. Both values are initialized to 20.

`TA_SEP` (`int`) two values. Allows you to specify x and y separation values between top/left corners of the bit maps for placing multiple areas without having to specify the x and y values for each. The separation is measured relative to the top left corner of the bitmaps. Any time this parameter is specified, its value becomes the default. Both values are initialized to 20. Note that the areas will proceed in the *x* direction first, then *y*.

`TA_TYPE` (`int`) mode for the touch area:
`TOUCH_HIDDEN`: display does not invert on press
`TOUCH_INVERT`: display inverts on press (default mode)
`TOUCH_TYPEMATIC`: will use the current typematic setup.

Any time this parameter is specified, its value becomes the default. The value is initialized to `TOUCH_INVERT`.

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): illegal coordinate
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_DispBitmap

```
int lcd_DispBitmap( int BmpID, int xCoord, int yCoord );
```

DESCRIPTION

Displays a bitmap on the Reach GTT. The bitmaps included with the Color Touchscreen Application Kit were drawn with the Windows Paint program. See the Reach manual (*SLCD Graphics Touch Terminal Technical Manual*) for details on downloading bitmaps.

PARAMETERS

BmpID	bitmap number
xCoord	starting *x* coordinate (0 = left side)
yCoord	starting *y* coordinate (0 = top)

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_DispText

```
int lcd_DispText( char *Text, int xCoord, int yCoord, int mode );
```

DESCRIPTION

Diaplay text on the Reach GTT. There are several display modes. To see their differences, run the sample program `\Samples\ColorTouchscreen\lcd_text.c` substituting other display modes for `MODE_NORMAL`.

PARAMETERS

`Text` Pointer to the text to display. The maximum number of characters is 126.

`xCoord` horizontal starting point. If both horizontal and vertical coordinates are 0, no coordinates are sent and the text will be displayed at a previously defined location.

`yCoord` vertical starting point. If both horizontal and vertical coordinates are 0, no coordinates are sent and the text will be displayed at a previously defined location.

`mode` One of these display modes:

- `MODE_NORMAL`
- `MODE_TRANS` - transparent, text is written on top of current display
- `MODE_XOR` - exclusive OR
- `MODE_REV` - foreground/background colors are reversed
- `MODE_TRANS_REV` - transparent reversed

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): string too long
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_DrawMode

```
int lcd_DrawMode( int mode );
```

DESCRIPTION

Sets the drawing mode.

PARAMETERS

mode 0 = normal
1 = XOR

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal mode
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_Font

```
int lcd_Font( char *Font );
```

DESCRIPTION

Sets the default font.

PARAMETER

Font Pointer to the font descriptor.

Current proportional fonts are:
"8", "10", "10S", "13", "13B", "16", "16B", "18BC", "24", "24B", "24BC", "32", "32B"

Current mono-spaced fonts are:
"4x6", "6x8", "6x9", "8x8", "8x9", "8x10", "8x12", "8x13", "8x15B", "8x16", "8x16L", "14x24", "16x32", "16x32i", "24x48", "32x64", "40x80", "60x120"

See the Reach Technology manual on the Reach Technology CD for additional details.

RETURN VALUE

lcd_SUCCESS: full message received
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_GetTouch

```
int lcd_GetTouch( int tdly );
```

DESCRIPTION

Recognizes a touchscreen press by examining the first character of the string sent by the Reach GTT. The first character will be “x,” “r” or “s.” This function returns a 16-bit integer, with the touchscreen area number in the low byte. For latched buttons, the high byte is 1 if the button was pressed and a 0 if not.

PARAMETER

tdly maximum number of milliseconds to wait

RETURN VALUE

`>0`: button number (0-127) or touchscreen area number (128-255)
`-1`: no touchscreen press detected

lcd_Line

```
int lcd_Line( int x0, int y0, int x1, int y1 );
```

DESCRIPTION

Draws a line using the current foreground color and pen width.

PARAMETERS

x0 starting pixel number of the x coordinate

y0 starting pixel number of the y coordinate

x1 ending pixel number of the x coordinate

y1 ending pixel number of the y coordinate

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Origin

```
int lcd_Origin( int xval, int yval );
```

DESCRIPTION

Sets the origin for subsequent commands.

PARAMETERS

xval *x* value (0–319)

yval *y* value (0–2390)

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal coordinate
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_PenWidth

```
int lcd_PenWidth( int width );
```

DESCRIPTION

Sets the width in pixels for lines, rectangles, and triangles, but not circles. The value is saved in global variable lcd_penwidth.

PARAMETERS

width width in pixels (1–200)

RETURN VALUE

lcd_SUCCESS: full message received
lcd_CMD_ERR-(parameter number): illegal pen width
lcd_UNKNOWN_ERR: packet too long
lcd_TIMEOUT_ERR: timeout error

lcd_Rectangle

```
int lcd_Rectangle( int x0, int y0, int x1, int y1, int style );
```

DESCRIPTION

Draws a rectangle using the current foreground color and pen width.

PARAMETERS

x0	top left x coordinate in pixels
y0	top left y coordinate in pixels
x1	bottom right x coordinate in pixels
y1	bottom right y coordinate in pixels
style	0 = regular line 1 = filled rectangle with current foreground color 2 = dotted rectangle (one-pixel dots—overrides pen width)

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_SetState

```
int lcd_SetState( int button, int state );
```

DESCRIPTION

Sets the state of a latching button.

PARAMETERS

button	The button number (1–127)
state	State value for the button: 0 (button is off) 1 (button is on)

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_CMD_ERR-`(parameter number): illegal button
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Triangle

```
int lcd_Triangle( int x0, int y0, int x1, int y1, int x2, int y2,
   int color );
```

DESCRIPTION

Draws a triangle using the current foreground color and pen width, unless a fill color is specified.

PARAMETERS

x0	top left vertex x coordinate in pixels
y0	top left vertex y coordinate in pixels
x1	top right vertex x coordinate in pixels
y1	top right vertex y coordinate in pixels
x2	bottom vertex x coordinate in pixels
y2	bottom vertex y coordinate in pixels
color	fill color in RGB format (see the section titled “Set Color (Detailed)” in the *SLCD Graphics Touch Terminal Technical Manual*) -1 = no fill color

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

lcd_Typematic

```
int lcd_Typematic( int delay, int repeatdelay );
```

DESCRIPTION

Sets typematic parameters for momentary buttons.

PARAMETERS

delay The delay in milliseconds before the button starts to repeat. Must be a multiple of 10 ms.

repeatdelay The delay in milliseconds between repeats. Must be a multiple of 10 ms.

RETURN VALUE

`lcd_SUCCESS`: full message received
`lcd_UNKNOWN_ERR`: packet too long
`lcd_TIMEOUT_ERR`: timeout error

Calculator Keypad Function Calls

The library SAMPLES\LIB\ColorTouchscreen\REACH_CALCULATOR.LIB provides the function calls to demonstrate a virtual calculator on a Reach GTT.

---

lcd_calculator_Display

```
int lcd_calculator_Display( int Keypad_BitMap_number, int X, int Y );
```

DESCRIPTION

Sets up and displays the specified keypad.

PARAMETERS

Keypad_BitMap_number	the bitmap number for the keypad
X	horizontal offset in pixels from the top left corner of the GTT
Y	vertical offset in pixels from the top left corner of the GTT

RETURN VALUE

lcd_SUCCESS

Calculator

```
float Calculator( void );
```

DESCRIPTION

Allows the operator to use the calculator displayed by lcd_calculator_Display(). This is a blocking function and will not return a value to the calling program until the operator presses the return key (RET).

RETURN VALUE

The result of the calculation.