SLCD Graphics Touch Terminal
Technical Manual

May 30, 2006

Reach Technology, Inc.
www.reachtech.com
(503) 675-6464
sales@reachtech.com

WARRANTY

Reach Technology Inc. ("Reach") warrants this product to be free from electrical and mechanical defects in materials and workmanship for a period of one year from the date of purchase. This warranty does not apply to defects in the Products caused by abuse, misuse, accident, casualty, alteration, negligence, repair not authorized by Reach, use on current or voltages other than specified herein, or application or installation not in accordance with published instruction manuals. This warranty is in lieu of any other warranty either expressed or implied.

Reach liability is limited to the repair or replacement of the Product only, and not costs of installation, removal, or damage to user's property or other liabilities. If Reach is unable to repair or replace a nonconforming Product, it may offer a refund of the amount paid to Reach for such Product in full satisfaction of its warranty obligation. Maximum liability of Reach is the cost of the Product.

Information furnished by Reach is believed to be accurate and reliable. However, no responsibility is assumed for the use of this information, nor for any infringements of patents or other rights of third parties which may result from its use. Reach retains the right to revise or change its products and documentation at any time without notice.

FCC Notice

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions:

(1) This device may not cause harmful interference, and

(2) this device must accept any interference received, including interference that may cause undesired operation.

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense. The user is cautioned that changes and modifications made to the equipment without approval of the manufacturer could void the user's authority to operate this equipment.

CE Certification

This device has been tested and complies with the appropriate standards required for the CE mark.

Table of Contents

Figures

1. Introduction

1.1. *Overview*

The SLCD Graphic Touch Terminal is a programmable operator interface for embedded systems. It enables an 8 or 16 bit microcontroller based system to have a modern graphical user interface. It runs on 12-24VDC and comes with both RS-232 and RS485/422 serial interfaces. It in enclosed in a metal case with a NEMA 4 compatible bezel and when mounted in a sealed case can provide complete waterproof operation.

1.2. *Features*

- Rugged, industrial enclosure
- Audible beeper for audible touch feedback and alarms
- Supports both CSTN and TFT displays
- RS-232 and RS-485/422 interfaces up to 115200 baud
- User downloadable bitmaps with RLE compression (512Kb of flash memory)
- Backlight enable and brightness control
- Available in portrait mode

1.3. *Specifications - Hitachi TFT Model 42-0080-01*

Power supply	12VDC ±10%, 650mA 24VDC ±10%, 350mA
Environmental	Temperature: -10°C to +70°C Operating: -30°C to +80°C Storage, max 48 hours Humidity: If ambient < 40°C, RH 85% max; otherwise must be lower than the humidity at 40°C and RH 85% Vibration, shock: see panel specification
Weight	2.65 lb / 1.2 kg
Enclosure	Painted steel bezel, yellow chromate inner mount and case. Bezel provides NEMA4 class waterproof capability when properly mounted on a flat plate.
Touchscreen	Four wire resistive type
Backlight	50,000 hour typical to half initial brightness 280 cd/m² typical brightness at mid-range of backlight brightness control
Interface	
	Serial RS232, ESD protected transceiver Serial RS422, ESD protected transceiver Serial RS485, full or half duplex
Certifications	FCC Part 15, Class A, CE Mark CISPR22, CNS13438, EN55022, ICES-003, VCCI

2. Dimensions

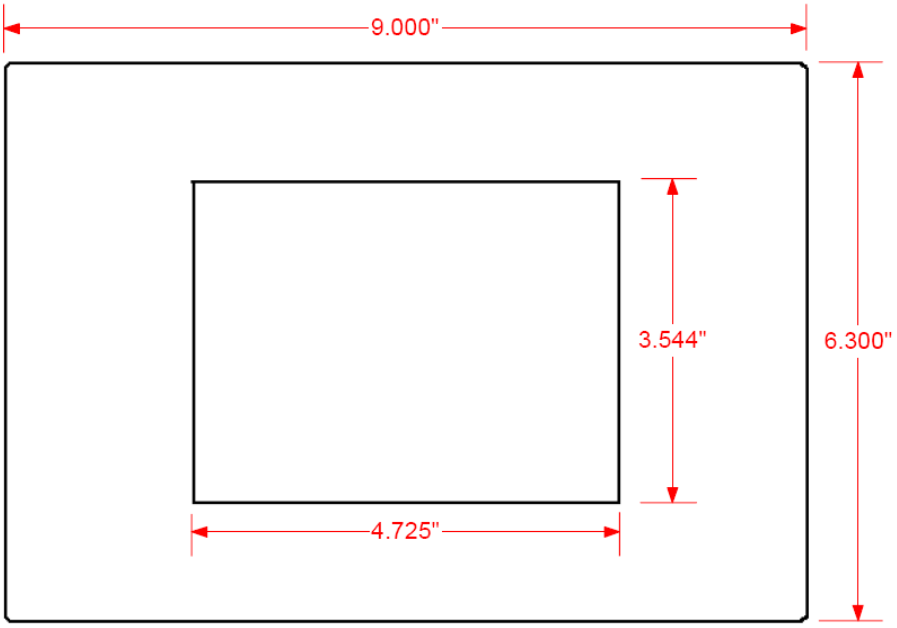


Figure 1: Front Bezel Dimensions (inches)

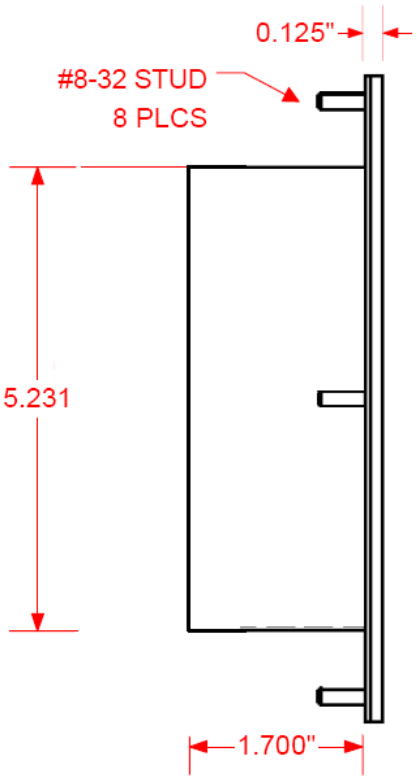


Figure 2: Side Dimensions (inches)

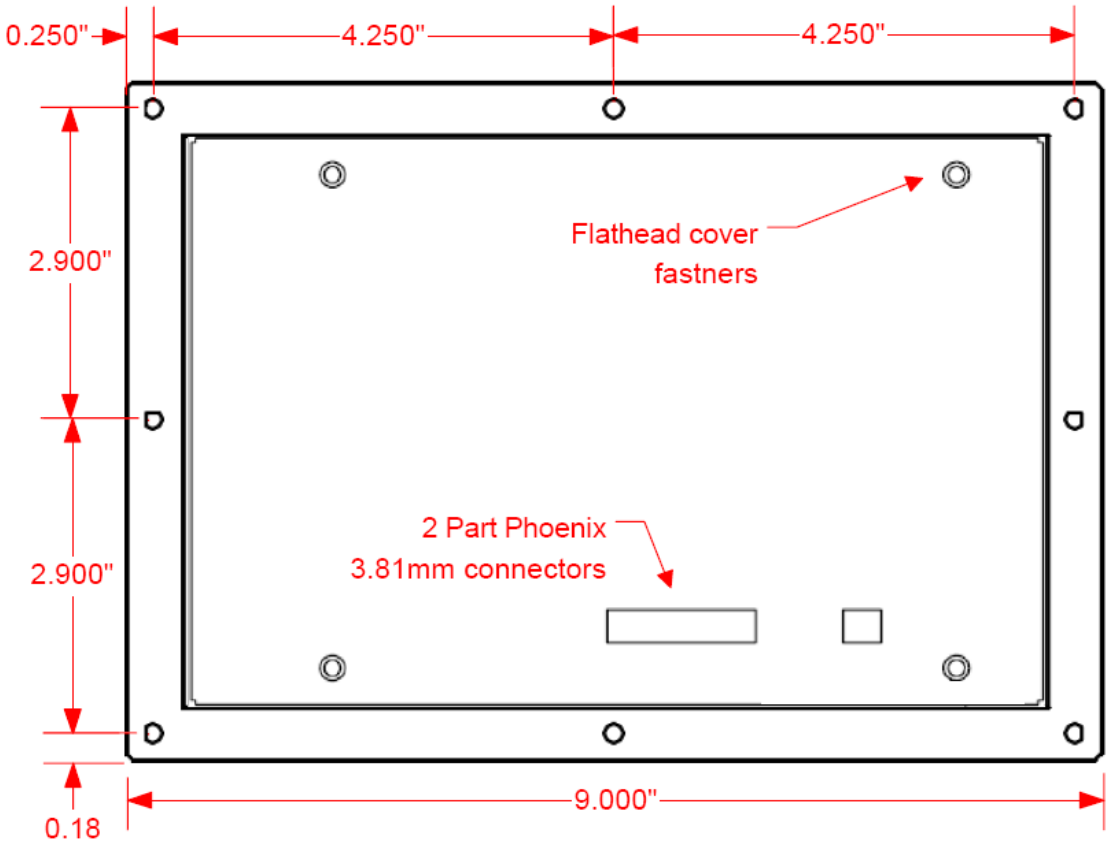


Figure 3: Rear Dimensions (inches)

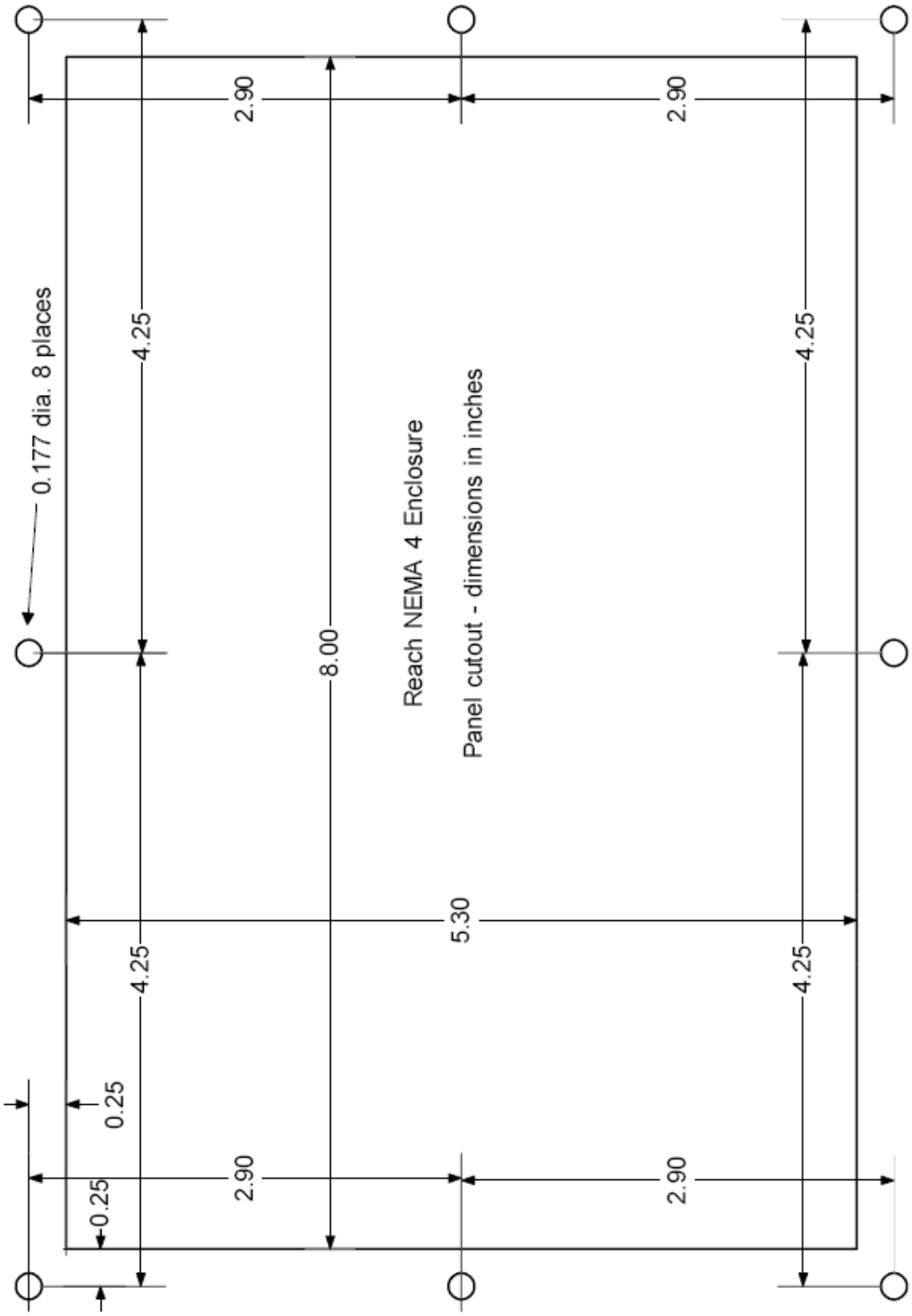


Figure 4: Panel cutout (not to scale)

3. Connectors and Power

3.1. *Serial Communications*

There are two connectors on the back of the unit for serial communications. These are two part screw type connectors with 3.81mm (0.15") spacing. Their pin assignments are shown in Figure 5.

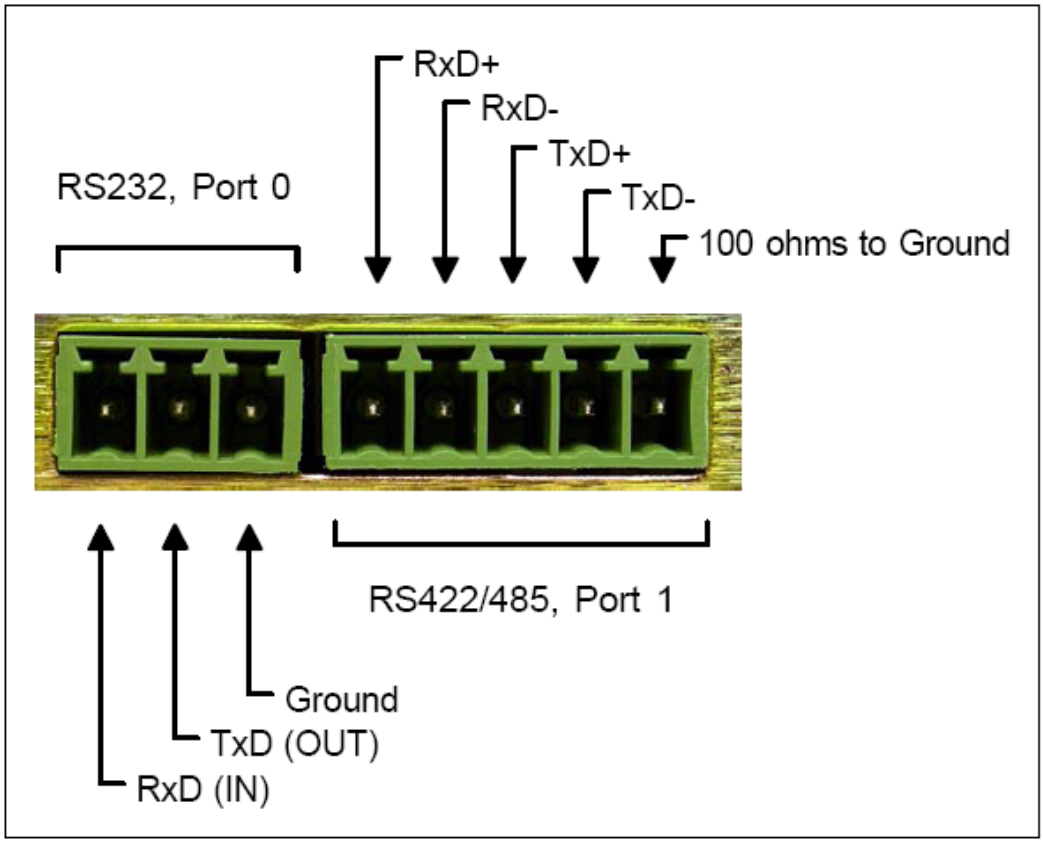


Figure 5: Back Panel Signal Connectors

3.2. *Power*

The power connector is a two part two pin connector similar to the serial connector. It is reverse polarity and overvoltage (surge) protected. Power is 12V - 24VDC.

3.3. *Connector Specifications*

Touch Terminal Connector Specifications:

Male connector	Phoenix Contact P/N	Phoenix Description
Serial 232 (3 pin)	1803439	MCV 1.5/3-G-3.81
Serial RS422/485 (5 pin)	1803455	MCV 1.5/5-G-3.81
Power (2 pin)	1803426	MCV 1.5/2-G-3.81

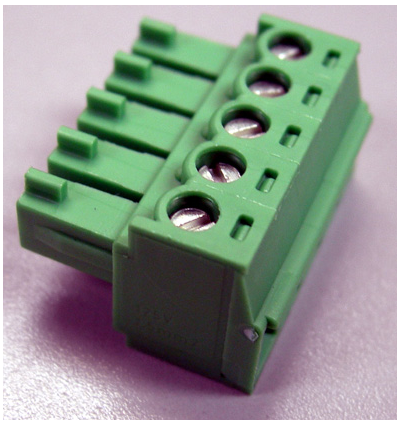


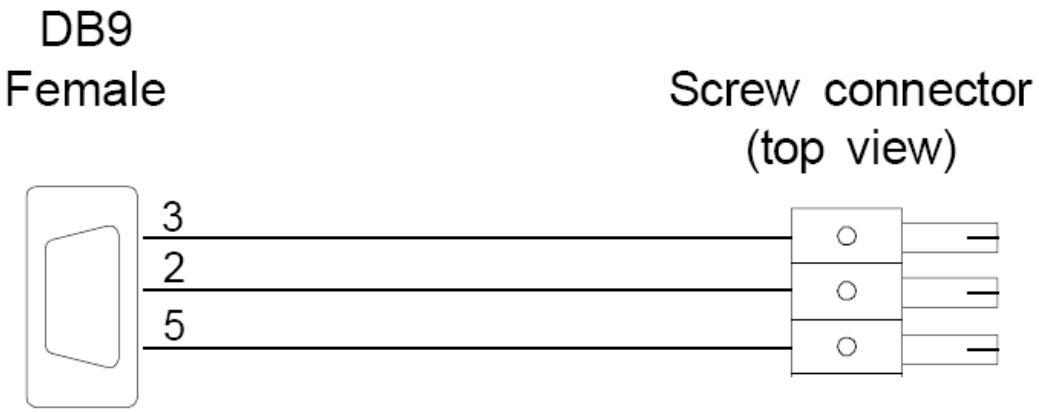
Figure 6: Mating screw terminal connector (5 pin version)

Mating Connector Specifications:

Female Screw connector	Phoenix Contact P/N	Phoenix Description
Serial 232 (3 pin)	1803439	MC 1.5/3-ST-3.81
Serial RS422/485 (5 pin)	1803455	MC 1.5/5-ST-3.81
Power (2 pin)	1803426	MC 1.5/2-ST-3.81

3.4. *DB9 to RS232 cable*

Reach provides an optional serial cable that has a DB9 female on one end and a three pin screw terminal mating connector on the other. This cable is designed to be connected one-to-one to the male DB9 serial port on a PC. See Appendix A for ordering information. The wiring diagram of this cable is shown below:



4. System Overview

4.1. *General*

The Graphic Touch Terminal (GTT) acts as a smart terminal and is typically connected to a "host" processor that implements the desired Graphical User Interface (GUI) by issuing commands to the GTT and processing button press responses from the GTT. In this manual, the term host is used to describe the device that connects to the GTT.

The unit has two serial ports so that either RS232 or RS422/485 communications can be used, however only one of these can be active at any time.

Note that it is possible to use the GTT as a host in a limited way by using macros and the OUTPUT command.

The unit contains flash memory that is used for bitmap and macro storage. A bitmap is equivalent to a Windows ™ bitmap file – it is a square image. Appendix D describes bitmaps and the BMPload program used to store these into the unit. A macro is a stored sequence of commands that can be executed from the host. This functionality is described in Appendix E.

4.2. *Communications Interface*

General

- Default communication is at a baud rate of 115200 with no parity, software (XON/XOFF) flow control, 8 bits of data, and 1 stop bit.
- The unit has two serial ports, but only one is active as the control port at any time. Commands are provided to designate which one is the "power on" control port. An auto-switch feature allows either port to temporarily take over control of the unit.
- The baud rate can be set to a different initial value on power-on by using the POWER-ON MACRO feature.
- ASCII commands consist of a command (one or more ASCII characters) followed by the data associated with that command, followed by a carriage return. In this manual, the return character (value 0x0D, decimal 13) is signified by <return>.
- Binary commands consist of series of hex bytes. The general format is as follows, where each <…> descriptor is a single byte. Note that the first byte indicates the command length and there is no trailing <return>.

  <0x80+number of bytes to follow><command byte><data0><data1>…<data n>

- Screen pixel values start at the upper left-hand corner. This is point x=0, y=0. The lower right corner is point x=319, y=239 (landscape mode).
- The maximum length of any command including the termination character is 127 characters.

Compressed Command Syntax

- All ASCII commands are shown with a space after the command mnemonic, for example:

  `p <pixels>`

  This command sets the line drawing width. This space is optional in all commands where the first argument is numeric (e.g. not text display) and can be removed to reduce code space and transmission overhead. For example.

  `p2<return>`

  sets the line width to 2.

4.3. *Input Buffer Processing*

Input Buffer

The unit has a nominal 512 byte input circular buffer. As commands are received, they are queued in the buffer and executed first come first served. After a command has been processed, the unit issues a "prompt" character followed by a <return> indicating the success or failure of the command. The '>' prompt indicates success and the '!' prompt indicates failure. Failure can be due to either a syntax error or an out-of-bounds parameter. Depending on how long a command takes to execute, one or more commands may be stacked in the input buffer. The unit will issue a prompt for each command after it executes. These prompts may be issued while the host is sending a command to the unit (full duplex operation).

The purpose of the circular buffer is to provide overlapped command issue and execution with full duplex communication. If this is not needed, the host can wait for the prompt before sending another command.

The unit issues a prompt when it has finished processing a command. This includes the null command which is just a <return>.

There is no special "power-on" prompt supplied when the unit first powers on. To detect that the board is available for commands, the host should send a null command (single <return> character) and wait at least 10ms for a success prompt back. Alternatively the POWER-ON MACRO command / feature can be used together with the OUTPUT command to send a unique message indicating that the unit is up and running.

Flow Control

The unit implements software flow control using the XON (decimal 17) and XOFF (decimal 19) characters. When the circular buffer is approximately ¾ full, an XOFF is issued to the host. An XON is then issued when the buffer is approximately ¼ full. If the host cannot, or does not want to accommodate software flow control, the host can make sure that no more than 2 commands are outstanding at any time. Given that the maximum length of any command is 127 bytes, this guarantees that the host will not be sent an XOFF character.

Buffer Limit Discussion

The input buffer can become full and unable to accept more data in two scenarios, both of which should never happen in normal operation. This discussion is presented because buffer overflow issues have presented security and reliability problems in PC and internet devices. The two scenarios are as follows. In both cases, the buffer limit event happens when the buffer is full and one more character is received and has to be thrown away.

Scenario #1: The host sends data that a) does not conform to the command specification, and b) keeps doing so until the buffer size limit is reached, and c) ignores the XOFF request from the unit. ASCII commands are limited to a total of 127 characters including the <return>. Input buffer limit will occur when enough data is sent without a <return> to fill the buffer. This indicates a flaw in the host protocol or a hardware failure (for example, the communication line is chattering).

Scenario #2: The host sends valid commands that take a long time to execute and ignores the XOFF request from the unit. The limit event can occur when the buffer is full of unexecuted commands.

In both of the above cases, when the unit detects a buffer limit it does the following:

- Discards the received character that caused the limit event, and resets (flushes) the entire input buffer. This is done in an attempt to make the error obvious to the GUI user. If a buffer overflow occurs it is a serious system error.

- Sends an overflow prompt to the host. The overflow prompt is '^'<return>. That is, shift-6 or caret followed by a return.

- Sends an XON character to the host (matches the XOFF that was previously sent)

Prompt Summary

The unit can issue the following prompts. These are in addition to any result of a command or button press event.

- '>'<return> Indicates the a command has been executed successfully

- '!'<return> Indicates that the command had a syntax or parameter error

- '^'<return> Indicates that an input buffer full event occurred.

- '?'<return> Indicates that a transmission line error occurred. This includes parity, framing, and receive overrun errors

4.4. *Touch interface*

The SLCD contains a touch controller that interfaces to a four wire resistive touchscreen. Touch sensitive areas of the display are defined as either "hotspots" or "buttons". When either of these is pressed or released, the SLCD can either notify the host directly or execute a "macro", or both. A macro is a predefined sequence of SLCD commands.

Hotspot

A hotspot is an area of the display that is touch sensitive. There are two types of hotspots – visible and invisible. A visible hotspot is the standard type and when touched, the display area of the hotspot is color inverted (technically XOR'd with the foreground color) to provide a visual indication that a hotspot has been activated. An invisible hotspot does not provide any visual indication when touched.

The invisible hotspot is useful where a touch control is used to switch display screens. If a visible hotspot is used, and the host redraws the screen when the hotspot is pressed, the hotspot area can become inverted when the user removes their finger from the screen.

Button

A button is a touch sensitive area that has two bitmaps associated with it. These bitmaps correspond to the two states of the button – 1) normal /not pressed and 2) active / pressed. This allows a button to look like any GUI object including pushbuttons, toggle switches, radio buttons, check boxes, and so forth.

There are two major types of buttons: normal (momentary) and latching. A momentary button changes visual state only when pressed. This is like a momentary pushbutton or a keyboard key. A latching button is like a checkbox – press and release it once and the checkbox is filled, press and release again to clear it.

Host Notification

When a touch sensitive area is pressed or released, the SLCD can either notify the host, execute a macro or both. See the BUTTON DEFINE and TOUCH MACRO ASSIGN commands for details.

4.5. *Host input processing*

When integrated into a host environment, the SLCD sends prompts, touch activity notifications, and user-defined text to the host it is connected to. In general, all SLCD messages are terminated with a <return>.

There can be no guarantee as to the order of arrival for prompts, touch notifications, etc. It is guaranteed that the messages arrive complete and do not overwrite each other. The debounce algorithm for touch processing ensures that the host is not overwhelmed by touch notifications.

4.6. *Control Port Autoswitch*

The Graphic Terminal unit has two serial ports - one RS232 and one RS422/485. Only one port is active at a time as the unit's control port. However in certain circumstances it is useful to be able to switch ports temporarily. This can be done by sending three consecutive <return> characters to the inactive port. Once this is done, the inactive port will become the active port temporarily.

5. Software Command Reference

Note: all command descriptions assume the display is running in landscape mode. X and Y parameter limits need to be swapped for portrait mode.

SET PEN WIDTH

Description	Sets the pen width for line drawing commands including line, rectangle *but not circle*. Default is width of 2.
Command:	`p <pixels>`
Arguments:	`<pixels>` is a number from 1 to 200.
Example:	`p 1` This sets the pen width to 1 pixel wide

SET DRAW MODE

Description	Sets the drawing mode for all line draw commands including draw line, rectangle, and circle. Note that for color displays the XOR mode produces the inverted RGB color.
Command:	`d [n|x]`
Arguments:	n: Normal drawing mode; draws with the colors from Set Color x: XOR drawing mode; inverts the existing pixel to draw lines.
Example:	`d n` This sets the drawing mode to normal

SET ORIGIN

Description:	Sets the origin for all subsequent operations including lines, text, bitmaps, buttons and so forth. This is useful for macros that draw compound objects. If the macro draws everything relative to (0 ,0), by setting the origin before calling the macro, the compound object can be placed anywhere on the screen. Note that the SET CURSOR command location is relative to this global origin.
Command:	o `<x> <y>`
Arguments:	`<x>` X axis value between 0 and 319 `<y>` Y axis value between 0 and 239

Example:	`o 10 20<return>`
	`t "hello" 0 0<return>`
	This sets the origin to x=10, y=20, and then displays the text "hello" at absolute location 10, 20

SET COLOR (basic)

Description — Sets the background and foreground color for all commands using a basic color palette.

Command — `s <fore> <back>`

Arguments: — <fore> = foreground color value per the table below

<back> = background color value per the table below

Color value	Color		Color value	Color
0	Black		9*	Grey
1	White		10*	Light Grey
2*	Blue		11*	Light Blue
3*	Green		12*	Light Green
4*	Cyan		13*	Light Cyan
5*	Red		14*	Light Red
6*	Magenta		15*	Light Magenta
7*	Brown		16*	Yellow
8*	Dark Grey			

* Only valid for color display

Example: — `s 0 1`

Form this point on, all objects will be drawn in black with a white background if applicable.

NOTE: — To reset the background after changing the color, the screen can be cleared using the command, '`z`'.

SET COLOR (detailed)

Description	Sets the background and foreground color for all commands using arbitrary RBG values.
Command	`S` <fore_detail> <back_detail>
Arguments:	<fore_detail> = foreground color value in RGB format
	<back_detail> = foreground color value in RGB format
	RGBformat = RGB where R, G, B are each a single character from 0 to f.
Example:	`S F00 069`
	Foreground = maximum red, background = minimum green, + half intensity blue
NOTE:	To reset the background after changing the color, the screen must be cleared using the command, '`z`'.
NOTE:	The SLCD has a fixed 8 bit palette which is expanded into 12 bit color. There are 16 shades of gray and 6 shades of each color. Therefore, not all of the 12 bit colors represented by the RGB argument can be shown. The discrete colors available are as follows:
	Gray scale: RGB = 000, 111, … EEE, FFF
	Color: R/G/B is either 0, 3, 6, 9, C, or F
	24 bit color space: for equivalent colors, duplicate the R/G/B value in both upper and lower hex nibble. Example: RGB = 069 is the same as color R=0x00, G=0x66, B=0x99.

SET FONT

Description	Sets the font to be used in subsequent TEXT DISPLAY commands.
Command:	`f` <type>
Arguments:	Proportional fonts: <type> = 8, 10, 10S, 13, 13B, 16, 16B, 18BC, 24, 24B, 24BC, 32, 32B
	Fixed width fonts: <type> = 4x6, 6x8, 6x9, 8x8, 8x9, 8x10, 8x12, 8x13, 8x15B, 8x16, 8x16L, 12x24,14x24, 16x32, 16x32i,
	Fixed width, symbol and CAPITALS only fonts: <type> = 24x48,32x64, 40x80, 60x120
	Where S-short, B-bold, C-comic, L=light (numbers only). For a complete description of each font their character sets, see Appendix A: Fonts.
Example:	`f 13B`
	Sets the current font to 13 point bold.

DISPLAY DOWNLOADED BITMAP IMAGE

Description:	Copies previously stored bitmap onto the screen at x y (top left corner of bitmap target) The Windows program BMPload.exe is used to download bitmaps into the SLCD external flash memory. See Appendix D for details.
Command:	`xi <number> x y`
Arguments:	<number> is bitmap number as listed in the "ls" command.
Example	`xi 4 10 20` This displays the 4th memory record at location (10,20).

LIST DOWNLOADED RECORDS

Description	Returns a summary of the contents of downloadable flash memory. This includes macros and downloaded bitmaps. This is for human debugging and the format is subject to change.
Command:	`ls`

LIST BITMAPS DETAIL

Description	Returns extended details of the bitmaps stored in downloadable flash memory. This is for human debugging and the format is subject to change.
Command:	`lsbmp`

TEXT DISPLAY

Description: Displays text string starting at a specified point using the currently set font. Draws text in foreground color inside a background color box unless options are specified. The backslash ("\") is the escape character, used to create double quotes ("\""), newline characters ("\n"), backslashes ("\\"), or arbitrary characters ("\xhh). A newline will move the next character down one line in the implied box starting at the x pixel location

Command:

```
t "text string" x y [R|T|X|TR]
```

or

```
t "text string"
```

Arguments: x is the left edge of the first character areas.

y is the top edge of the first character area.

R – Reverse: foreground / background colors are reversed.

T – Transparent: text written on top of current display with no "background box".

X – XOR

TR – Transparent reversed

Note: Quotes are required around the text string. ***The entire command including <return> must be less than 120 characters.***

If only the text string is provided as an argument, the text is written to the current cursor position, and the last mode (R, T, X, TR) specified is maintained. See SET CURSOR command.

Examples:

```
t "Press \"next\" \nto continue" 10 0
```

This puts the text

```
Press "next"
to continue
```

With the top left corner of the ‘P’ at location x=10, y=0

```
t "\xa9Copyright" 0 0
```

displays the text

```
©Copyright
```

at the top left corner of the screen

SET CURSOR

Description: Sets the location where text will be displayed by default. This is used with the TEXT DISPLAY command where only the text to be displayed is the argument. This is useful when text is generated by a macro and the location is specified before the macro is invoked.

Command: sc `x y`

Example: sc 10 20<return>
t "hello"

Is equivalent to:

t "hello" 10 20

DRAW LINE

Description: Draws a line from (x0,y0) to (x1,y1) using the foreground color.

Command: `l x0 y0 x1 y1`

Example: `l 0 0 319 239`

This will draw a line from the upper left-hand corner of the screen to the lower right hand corner

DRAW RECTANGLE

Description: Draws a rectangle using the foreground color or an arbitrary color

Command: `r x0 y0 x1 y1` [style] [color]

Arguments: Upper left corner is (x0,y0) and lower right corner at (x1,y1).

style: omitted=regular line, 1=filled, 2= one pixel wide dotted line.

color: fill color in RGB format (see SET COLOR detailed)

Example: `r 100 100 180 120`

Draws a rectangle positioned at 100,100 with a width of 80 and a height of 20.

`r 100 100 180 120 1`

Draws a rectangle filled with the foreground color positioned at 100,100 with a width of 80 and a height of 20.

`r 50 100 180 120 1 C03`

Draws a rectangle filled with the color R=C,G=0,B=3 positioned at 50,100 with a width of 80 and a height of 20

DRAW CIRCLE

Description:	Draws a single pixel width circle using the foreground color. If the optional fill argument is supplied, the entire circle is filled with the foreground color.
Command:	`c x0 y0 r [f]`
Arguments:	Center is (x0,y0) with radius r. The circle is not filled if f is omitted, and filled if f=1.

Example:	`c 100 100 50`
	Draws a circle centered at 100,100 with a radius of 50.
	`c 100 100 50 1`
	Draws a circle filled with the foreground color centered at 100, 100 with a radius of 50.

DRAW TRIANGLE

Description:	Draws a triangle using the current pen width and foreground color for the line. If the optional fill argument is supplied, the triangle is also filled with the specified color. Note: to fill without a outline border, set the pen width to 1.
Command:	`tr x0 y0 x1 y1 x2 y2` [RGB]
Arguments:	The three x, y sets are the triangle vertices. The optional color fill argument is three hex characters; see SET COLOR DETAILED command.

Example:	`tr 10 10 10 100 200 200`
	Draws a triangle with points (10,10), (10,100), (100,200).
	`tr 10 10 10 100 200 200 0CC`
	Same as above, but the triangle is filled with light blue

CHART DEFINE

Description: Defines a chart to which data can be added. See CHART VALUE command to add data to a chart. If more data points are added than can fit on the graph, the data starts again on the left in "Oscilloscope" style.

Command: `cd n x0 y0 x1 y1 t dw bv tv bc <pens>`

Arguments:

`n` - chart index from 0 to 9 (maximum 10 charts).

`x0, y0` and `x1, y1` are the top left corner and bottom right corners of the chart area

`t` - chart type; must be 1

`dw` - data width, number of pixels horizontally between chart data points

`bv` - bottom data value (lowest y value)

`tv` - top data value (highest y value)

`bv` - bottom data value (lowest y value)

`bc` - background color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)

`<pens>` - one or more sets of two values: pen width and pen color. Width is 1 or 2, color is same format as "bc" parameter.

Example: `cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00`

Defines a chart in the rectangular area (10,20), (110,120). Each data value will be 4 horizontal pixels wide. The chart (‘Y’) values are scaled from 0 to 99. The background color is dark gray (333). Two pens are defined: the first is pen width 2, color teal (0FF), the second is pen width 1, color red (F00).

CHART VALUES

Description: Adds data points to previously defined chart. Note: if multiple pens are defined, they are drawn in order first to last – if multiple pens have the same value only the last pen color will be visible.

Command: `cv n pen0_value [pen1_value ..]`

Arguments: `n` - chart index from 0 to 9 (maximum 10 charts).

`pen0_value` - value to be added for pen 0. Must be in the range previously defined for chart 'n'.

`pen1_value` - additional values for each pen defined for chart 'n'. Must be in the range previously defined for chart 'n'.

Example:

```
cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00
cv 0 30 50
cv 0 40 60
```

Defines a chart (see CHART DEFINE) and enters a value of 30 for the teal pen and 50 for the red pen. The lines will be 4 horizontal pixels long for each. Next two more data points are added.

LEVELBAR DEFINE

Description: Defines a "levelbar" object. The object provides scaling and different colors for different levels, similar to a sound level meter. Note that the object is not visible until a value is assigned – see the LEVELBAR VALUE command.

Command: `ld n x0 y0 x1 y1 or inv bv bc <levels>`

Arguments: `n` - object index from 0 to 9 (maximum 10 charts).

`x0, y0` and `x1, y1` are the top left corner and bottom right corners of the object's area

`or` - orientation: 0 = vertical, 1 = horizontal

`inv` - invert: 0 = no (low value at bottom / left); 1 = yes (low value at top / right)

`bv` - bottom data value; should be 1 if value 0 means no level displayed

`bc` - background color in RGB format (3 ASCII hex characters – see SET COLOR DETAILED)

`<levels>` - one or more sets of two values: value and associated color. These start with the maximum and go down. At most 3 sets are possible. Color is the same format at the `bc` parameter.

Example: `ld 0 10 10 30 200 0 0 1 333 99 F00 50 FF0 40 0F0`

Defines a levelbar in the rectangular area (10,10), (30,200). Levelbar is vertical with the lowest value at the bottom; minimum visible value of 1, with background color dark gray (333). Three color bands are defined: red (F00) from 99 to 51, yellow (FF0) from 50 to 41, and green (0F0) from 40 to 1.

LEVELBAR VALUE

Description:	Sets the value of a previously defined "levelbar" object.
Command:	`lv n val`
Arguments:	`n` - object index `val` - value for the levelbar.
Example:	`lv 0 50` Sets levelbar 0 to value 50..

CLEAR SCREEN

Description:	Clears the screen to the background color and removes any buttons and hotspots.
Command:	`z`

BLANK SCREEN (16 color)

Description:	While preserving the data and all buttons and hotspots, this command uses the Lookup Table to set the entire screen to one color. Note: this only works if just the basic colors have been used to draw on the screen.
Command:	`sb <color>`
Arguments:	<color> is 0 to 16 per the colors of the SET COLOR (basic) command.
Example:	`sb 12` Sets the entire screen to Light Green

UNBLANK SCREEN (16 color)

Description:	Reverses the effect of the blank screen (basic) command
Command:	`su`

BLANK SCREEN (complete)

Description:	While preserving the data and all buttons and hotspots, this command uses the Lookup Table to set the entire screen to one color. This command clears the entire Lookup Table to one color.
Command	`SB` <color_detail>
Arguments:	<color_detail> = foreground color value in RGB format
	RGBformat = RGB where R, G, B are each a single character from 0 to f.
Example:	`SB 113`
	Sets the entire screen to a light blue.

UNBLANK SCREEN (complete)

Description:	Reverses the effect of the blank screen (detailed) command by resetting the Lookup Table to the default palette.
Command:	`SU`

BUTTON DEFINE - MOMENTARY

Description: Defines a momentary touch button on the screen. When touched, the host is notified, and optionally a macro can be invoked – see TOUCH MACRO ASSIGN.

Note: when a button is number is redefined, all macro assignments are cleared.

Command: `bd <n> <x> <y> <type> "text" <dx> <dy> <bmp0> <bmp1>`

Arguments:

`<n>` Button number, must be in the range of 0 to 127.

`<x> <y>` Upper left hand corner of the button

`<type>` Button type:

- 1 Standard. Displays <bmp0> normally, and <bmp1> when pressed. Host is notified when button is pressed, but not when it is released.
- 3 Typematic. Same as regular but with typematic functionality; that is, host notification repeats after the button is held down. See SET TYPEMATIC PARAMETERS command.
- 4 Standard except host is notified only when the button is released.
- 5 Standard with both press and release notification.

`"text"` Text string to be displayed on the button. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10.

`<dx>` Text offset in the x direction from the upper left-hand corner of the button.

`<dy>` Text offset in the y direction from the upper left-hand corner of the button.

`<bmp0>` Index of bitmap displayed in the unpressed state.

`<bmp1>` Index of bitmap displayed in the pressed state.

Note: both bitmaps must be the same size.

Host notification, type 1, 3, or 5 when button pressed:

```
x<n><return>
```

Host notification, type 4, 5 when button released:

```
r<n><return>
```

BUTTON DEFINE – MOMENTARY (continued)

Example: `bd 23 150 100 1 "Test" 10 12 2 3`

Defines button number 23 displayed at x=150, y=100. The "un-pressed" image uses bitmap 2 with the text "Test" drawn on the bitmap in the current font at offset x=10, y=12 from the top left corner of the bitmap. The "pressed" image is the same except bitmap 3 is used. Bitmaps 2, 3 must be loaded and have the same size. When pressed, the host is sent:

`x23<return>`

Example: `bd 0 10 20 5 "" 0 0 5 6`

Defines button 0 displayed at x=10, y=20. The "un-pressed" image uses bitmap 5, and the "pressed" image uses bitmap 6. No text is supplied so the bitmaps themselves must contain the description. For example, the bitmap 5 could show a toggle switch in the "up" position, and bitmap6 could show a toggle switch in the "down" position.. Bitmaps 5, 6 must be loaded and have the same size. When pressed, the host is sent:

`x0<return>`

When released, the host is sent:

`r0<return>`

BUTTON DEFINE – LATCHING STATE

Description: Defines a touch button on the screen with two distinct states. This is the equivalent of a retractable pen actuator – push it down, it clicks and stays down; push it again and is comes back up. When touched, the host is notified. A macro can also be invoked from a button press – see TOUCH MACRO ASSIGN.

Command: `bd <n> <x> <y> <type> "text0" "text1" <dx0> <dy0> <dx1> <dy1> <bmp0> <bmp1>`

Arguments:

`<n>` Button number, must be in the range of 0 to 127.

`<x> <y>` Upper left hand corner of the button

`<type>` Button type:

- 2 Latching. Displays <bmp0> in state 0 and <bmp1> in state 1
- 20 Latching. Same as above. (Initial state is set to state 0)
- 21 Latching. Same as above, with initial state set to state 1

`"text0"` Text string to be displayed on the button in state 0. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10.

`"text1"` Text string to be displayed on the button in state 1. The current foreground color will be used for the text..

`<dx0>` Text offset in the x direction from the upper left-hand corner of the button for `"text0"`.

`<dy0>` Text offset in the y direction from the upper left-hand corner of the button for `"text0"`.

`<dx1>` Same as above for `"text1"`.

`<dy1>` Same as above for `"text1"`.

`<bmp0>` Index of bitmap displayed in state 0.

`<bmp1>` Index of bitmap displayed in the state 1.

Note: both bitmaps must be the same size.

Host notification: `s<n><s><return>` where `<s>` is 0 or 1 for the new state.

Example: `bd 3 20 30 2 "GO" "STOP" 10 5 3 5 7 8`

Define a latching button #3 at x=20, y=30 using bitmaps 7 and 8 with the text "GO" displayed in state 0 at offset (10,5) and "STOP" in state 1 at offset (3,5).

```
bd 3 20 30 2 "" "" 0 0 0 0 2 3
```

Define a button as above, but use bitmaps that have the GO and STOP text as part of the bitmaps so no text is needed.

SET (LATCHING) STATE BUTTON

Description:	Changes the latching state button to a specified state. This can be used to implement a set of selection buttons where pushing one down causes the others to pop up.
Command:	`ssb` <n> <state>
Arguments:	<n> - latching button number (0-127) <state> - specifies the desired state (0 or 1).
Example:	`ssb 5 1` This command would force a button defined with DEFINE BUTTON (type=2) into state 1.

BUTTON CLEAR

Description:	Clears the definition for the specified button. *Note: This DOES NOT CHANGE THE SCREEN IMAGE.*
Command:	`bc` <n>
Arguments:	<n> - previously defined button number (0-127)
Example:	`bc 3` This command clears the definition of the previously defined button 3.

DEFINE HOTSPOT (VISIBLE TOUCH AREA)

Description:	Define a touch area on the screen. When touched, this area's number will be returned on the serial control line. The area defined will be set to reverse video while touched.
Command:	`x` <n> `x0 y0 x1 y1`
Arguments:	<n> touch button number. Must be in the range of 128 to 255. (x0,y0), and (x1,y1) specify the touch area for this button.
Returns:	x<n><return> when the corresponding button is pushed. Note that once a button is defined, the return string can be transmitted at any time including during a command transmission to the unit (full duplex).
Example:	`x 135 100 100 180 140` Draws a rectangular hotspot with height of width of 80 and height of 40.

DEFINE SPECIAL HOTSPOT (INVISIBLE TOUCH AREA)

Description: Same as DEFINE HOTSPOT except that the touch area is not reverse video highlighted when touched. This allows a "hidden" touch area to be placed on the screen.

Command: `xs <n> x0 y0 x1 y1`

Arguments, Returns: same as DEFINE HOTSPOT command.

Example: `xs 135 100 100 180 140`

Draws a rectangular hotspot with height of width of 80 and height of 40.

DEFINE TYPEMATIC TOUCH AREA

Description: Same as DEFINE HOTSPOT except that the touch area is typematic and will repeatedly send the return code if the area is pressed continuously.

Command: `xt <n> x0 y0 x1 y1`

Arguments, Returns: same as DEFINE HOTSPOT command.

DISABLE TOUCH

Description: Temporarily disables touch area or button. Once disabled, the button graphic may be overwritten by a pop-up or other element. Disabled touch areas / buttons can be re-enabled.

Command: `xd <n>`

Arguments: <n> touch button number. Must be in the range of 0 to 255, and must have been previously defined.

Example: `xd 1`

Disables previously defined button 1.

ENABLE TOUCH

Description: Re-enables touch area or button. For buttons, the state of the button is remembered and the correct graphic is displayed.

Command: `xe <n>`

Arguments: <n> touch button number. Must be in the range of 0 to 255, and must have been previously defined.

Example: `xe 1`

Enables previously disabled button 1.

CLEAR TOUCH AREA

Description: Clears the previously defined touch area.

Command: `xc <n>`

CLEAR ALL TOUCH

Description: Clears all previously defined touch areas including the button touch areas.

Command: `xc all`

MACRO EXECUTE

Description: Runs a macro (list of commands) previously stored in flash memory. The BMPload.exe program is used to store both macros and bitmaps into the flash; see Appendix D. See Appendix E for the macro file format.

The stored macros can be defined to take arguments when called. In this case, the arguments are specified by this command. For more details on parameterized macros, see Appendix E.

NOTE: the maximum number of arguments, and maximum size of each argument is version-dependent. See Appendix E.

Command: `m <n> [macro parameters . . ]`

Arguments: <n> is the macro number between 1 and 255. If the macro takes arguments, the values are supplied in order after the macro number. They are delimited by spaces, If a space is to be included in an argument, the argument must be enclosed with double quotes.

Example: `m2`

This causes macro #2 to execute.

Example: `m 3 "" 2`

This causes macro #3 to execute with a value for the first parameter of a space character, and the value of the second parameter the number 2.

LIST MACROS DETAIL

Description Returns extended details of the macros stored in downloadable flash memory. This is for human debugging and the format is subject to change.

This command also lists the current button to macro assignments.

Command: `lsmac`

TOUCH MACRO ASSIGN

Description: Links a button or hotspot to a macro. When the button or hotspot is touched, the associated macro is executed. See the MACRO NOTIFY command for host notification of macro execution options.

Command: `xm <touch index><macro index> [<macro2 index>]`

Arguments: <touch index> is the index of the button or hotspot.

<macro index> is the index of the macro to be executed when the button or hotspot is pressed, or in the case of latching buttons, when the button is pressed to change from state 0 to state 1.

<macro2 index> is an optional parameter. In the case of button or hotspot, this specifies a macro to be executed when the touch area is released. For latching buttons, this macro is executed when the button changes state from state 1 to 0.

Examples:

```
xm 128 2
```

This will run macro #2 when hotspot 128 is pressed.

```
xm 128 2 3
```

This will run macro #2 when hotspot 128 is pressed, and #3 when it is released.

```
bd 2 150 100 2 "OFF" "ON" 30 10 30 10
xm 2 5 3
```

This creates a latching button and executes macro 5 when the button is switched to "ON" and macro 3 when the button is switched to "OFF"

TOUCH MACRO ASSIGN QUIET

Description: This has the same functionality as TOUCH MACRO ASSIGN except that the standard button response to the host is disabled AND pushing the button does not cause a beep. This is useful when the macro contains an OUT command to generate arbitrary button responses.

Command: `xmq <touch index><macro index> [<macro2 index>]`

Arguments: See TOUCH MACRO ASSIGN.

Example:

```
xmq 5 2
```

This will run macro #2 whenever button 5 is touched, and the standard button press response will not be given to the host.

TOUCH MACRO ASSIGN WITH PARAMETERS

Description: Links a button or hotspot to a parameterized macro. When the button or hotspot is touched, the associated macro is executed with the specified arguments.

NOTE: the maximum number of arguments, and maximum size of each argument is version-dependent. See Appendix E.

Command: `xa[q] <t><action><m><args>`

Arguments:

`<t>` the index of the button or hotspot.

`<action>` is one of:

- `p` - execute the macro and arguments when the button is pressed (momentary) or when it changes from state 0 to state 1 (latching).
- `1` - same as above.
- `r` - execute the macro and arguments when the button is released (momentary) or when it changes from state 1 to state 0 (latching).
- 0 - same as above.

`<m>` the index of the macro to be executed when the button or hotspot is pressed.

`<args>` arguments for the macro. These are delimited by spaces. Double quotes can be used to surround the argument if it contains spaces..

Example 1:

```
bd 1 100 100 1 "test" 10 15
xa 1 p 17 Check
```

This defines button 1 and assigns macro 17 to run with the first argument = Check when button 1 is pushed. The corresponding macro definition could look as follows:

```
#define test 17
t 0 0 `0`
#end
```

When button 1 is pushed, macro 17 is invoked and the following command will be executed:

```
t 0 0 Check
```

TOUCH MACRO ASSIGN WITH PARAMETERS (cont'd)

Example 2: bd `xa 1 p 17 Check`

Assuming button 1 has been defined in a previous command, this assigns macro 17 to run with the first argument Check when button 1 is pushed. The corresponding macro definition could look as follows:

```
#define test 17
t 0 0 `0`
#end
```

When button 1 is pushed, macro 17 is invoked and the following command will be executed:

```
t 0 0 Check
```

OUTPUT STRING

Description: This outputs a text string to the serial port. This is typically used in macros that are assigned to buttons using the quiet feature above. This enables a button press to output arbitrary text to the serial port.

Command: `out "<text string>"`

Arguments: The text string can contain the following escapes:

`\\` = \

`\"` = "

`\n` = line feed

`\r` = return

`\xhh` = arbitrary character with hex value hh

Example: `out "\x48ello \"world\"\r"`

This will send the following string out on the serial port:

```
Hello "world"<return>
```

SPLASH SCREEN

Description: Selects a downloaded bitmap as the power-on "splash screen". This takes the place of the initial display version text string.

The Windows program BMPload.exe is used to download bitmaps into the SLCD external flash memory. See Appendix D for details.

Command: `*SPL <number>`

Arguments: <number> is bitmap number as listed in the "ls" command. If 0 is used, no bitmap is selected and the standard product text string is displayed.

Example `*SPL 5`

This displays the 5th memory record at location (0, 0) on power-on reset.

SET TYPEMATIC PARAMETERS

Description:	Sets the delay and repeat rate for typematic buttons. These are stored in non-volatile memory.
Command:	`typematic <delay> <repeat>`
Arguments:	<delay> is the number of 10's of milliseconds a typematic button must be held down before it starts to repeat. <repeat> is the repeat interval in 10s of milliseconds.
Example:	`typematic 200 50` This sets the delay to 2 seconds and the repeat rate at 500ms = 2 per second.
Example return:	`Delay 2000ms, Repeat 500ms<return>`

SET TOUCH SWITCH DEBOUNCE

Description:	Sets the delay between touch button responses. This is stored in non-volatile memory. Manufacturing default is 100ms.
Command:	`*debounce <delay>`
Return:	`Debounce = ????ms<return>`
Arguments:	<delay> is the number of milliseconds after a touch is recognized that another touch can be recognized. If no argument is given, the current value is returned.
Example:	`*debounce 50` This sets the delay to 50 milliseconds.

RESET TOUCH CALIBRATION

Description:	Resets the touch calibration to a default value. Doing this before setting the entire screen to be a touch sensitive area guarantees that a touch will be seen independent of the current touch calibration.
Command:	`*RT`
Returns:	(standard prompt)

TOUCH CALIBRATE

Description:	Runs the touch calibration procedure. This displays calibration points on the screen and asks the user to touch them to calibrate the screen. Note that a command prompt is not given until the procedure has been completed. Calibration values are stored in non-volatile memory and restored on power-on.
Command:	`tc`
Returns:	(nothing)

BEEP ONCE

Description:	Beeps the beeper for <count> ms. This will temporarily interrupt any running repeating beep. A prompt is returned immediately even if the beep continues.
Command:	`beep` <count>
Arguments:	<count> is number of ms to sound the beeper.

BEEP WAIT

Description:	Beeps the beeper for <count> ms. This will temporarily interrupt any running repeating beep. The system issues a command prompt only after the beep has stopped.
Command:	`beepw` <count>
Arguments:	<count> is number of ms to sound the beeper.

BEEP VOLUME

Description:	Sets the volume level of the beeper. The value is stored in non-volatile memory and restored on power-on.
Command:	`bv [+|-]`<level>
Arguments:	<level> is number from 0 through 255. Default is 200. Loudest is 255. The optional '+' or '-' prefix changes the <level> into an increment up or down.

BEEP FREQUENCY

NOTE: the beep frequency is set at factory to generate maximum loudness level.

Description: Sets the frequency of the beeper. The value is stored in non-volatile memory and restored on power-on. This command is used during factory calibration to set the sound level as the sounders used resonate at slightly different frequencies. If you use it to change the frequency, the factory test results are invalid. Please do not use without premeditation! The *MFGRESET command cannot restore the original value of this setting.

Command: `bf [<hertz>]`

Arguments: <hertz> is number from 1 through 4000. Default is 2650, but may be slightly different due to volume calibration at the factory. If no argument is supplied, the current frequency is returned as a variable length decimal number.

Example `bf 2500`

Sets the beep frequency to 2500 Hertz

`bf`

Returns 2500 after the above command was issued.

BEEP REPEAT

Description: Beeps the beeper for <on> ms, stays silent for <off> ms, and then repeats until the values are changed with another "rb" command. Can be temporarily overridden by a regular "beep" command. If <on> and <off> are both 0 the repeat stops.

Command: `rb` <on> <off> [alarm]

Arguments: <on> is number of ms to sound the beeper.

<off> is number of ms to stay silent before beeping again.

[alarm] is an optional parameter to use the alarm sound instead of a steady tone. See alarm command for valid alarm numbers.

Example `rb 100 400`

Repeatedly beeps for 100 ms then goes silent for 400 ms during each 500 ms cycle.

BEEP TOUCH

Description:	Sets the duration of the audible feedback beep when a hotspot or button is pressed. Not stored in non-volatile memory. Default is 10 which equals 100ms beep.
Command:	`bb <number>`
Arguments:	<number> is tens of milliseconds to sound the beeper.
Example	`bb 10`
	Sets the beep feedback to power-on value.

ALARM

Description:	Sounds an alarm sound using the beeper.
Command:	`al <alarm> <count>`
Arguments:	<alarm> is the alarm sound: 1 = whoop 2 = annoy 3 = dee-dah <count> is number of ms to sound the beeper.
Example	`al 2 1500`
	Sounds the "annoy" alarm for 1.5 seconds.

WAIT

Description:	Returns command prompt after a specified number of milliseconds. This is useful in macros that implement self-paced demonstrations as it delays execution of the next line.
Command:	`w <number of milliseconds>`
Arguments:	`<number of milliseconds>` is the number of milliseconds to delay, maximum is 65535.
Example	`w 1000`
	This will return the command prompt in 1 second or in the case of a macro, delays execution by 1 second.

DISPLAY ON/OFF

Description: Turns power to the display (and backlight) on or off. This can be used to reduce power consumption. With passive STN or CSTN panels, it is highly recommended that the "v off" command be executed before power is removed fro the panel (unit is powered down). If this is not done, a horizontal line can be seen on the display when power is abruptly removed.

Command: `v <on|off>`

EXTERNAL BACKLIGHT ON/OFF

Description: Turns the external backlight control on or off via J10.

Command: `xbl <on|off>`

EXTERNAL BACKLIGHT BRIGHTNESS CONTROL

Description: Sets the brightness of the external backlight if the external unit supports this feature. The value is stored in non-volatile memory and restored on power-on.

Command: `xbb [+|-]<level>`

Arguments: <level> is number from 0 through 255. The 0 is the dimmest and 255 is brightest. The optional '+' or '-' prefix makes the level value an increment up or down rather than an absolute value. The value saturates at 0 and 255 without error; in other words if the level is at 255 and an "xbb +10" is issued, the level stays at 255 and no error prompt is issued.

Example: `xbb -10`

This will reduce the brightness by 10 units but no lower than 0.

SET BAUD RATE

Description: Sets a new baud rate for port specified. The baud0 command sets the MAIN port and the baud1 command sets the AUX port. This is temporary and the unit will revert to the default setting the next time power is cycled.

Command: `baud0 [115200|57600|38400|19200|9600]`

`Or`

`baud1 [115200|57600|38400|19200|9600]`

Argument: baud rate

Example: `baud0 57600(Sets baud rate of MAIN port to 57600)`

Example: `baud1 19200(Sets baud rate of AUX port to 19200)`

CONTRAST

Description: For passive panels only: changes the display contrast up or down. Note that the actual effect on the display may be reversed (e.g. up = less contrast) depending on the display used. The value is stored in non-volatile memory and restored on power-on

Command:

```
C+   Changes contrast one click up
C-   Changes contrast one click down
C=   Sets contrast to middle value
C>   Sets contrast to maximum value
C<   Sets contrast to minimum value
```

VERSION

Description: Displays the version of the software

Command: `vers`

DEMO

Description: Invokes the demo macro if valid. This is the same as if the TX and RX of the RS232 are connected together on power-up. Note that the command is case sensitive.

Command: `Demo`

WRITE LCD CONTROLLER

Description: Allows writes directly to the S1D13705 LCD controller. DO NOT USE UNLESS YOU HAVE THE 13705 MANUAL. Note that any argument that is 0 must be given as 00 or 0x0 (bug).

Command: `XW <hex register> <hex value>`

Returns: `LCD Reg xx <- xx<newline><return>`

READ LCD CONTROLLER

Description: Allows reads directly from the S1D13705 LCD controller. DO NOT USE UNLESS YOU HAVE THE 13705 MANUAL. Note that any argument that is 0 must be given as 00 or 0x0 (bug).

Command: `XR <hex register>`

Returns: `LCD Reg xx = xx<newline><return>`

READ FRAME BUFFER LINE

Description:	Displays 320 comma separated frame buffer hex bytes for a given display line. Each byte is a palette index.
Command:	`*FB <line>`
Arguments:	`<line>` is the display line buffer from 0 to 239.

CRC SCREEN

Description:	Returns the 16 bit CRC of the display buffer. This can be used to generate automated tests to verify correct user interface operation across a user's system software version changes.
Command:	`*CRC`
Returns:	0xXXXX<return> where XXXX is a hex number.

CRC EXTERNAL FLASH

Description:	Returns the 16 bit CRC of the external flash used to store macros and bitmaps. This can be used in production code to verify that the correct bitmaps are loaded in the board.
Command:	`*CEXT`
Returns:	0xXXXX<return> where XXXX is a hex number.

CRC PROCESSOR CODE

Description:	Returns a 16 bit CRC of the entire processor code space. The purpose is to verify the contents of code memory without doing a byte-by-byte comparison.
Command:	`*CSUM`
Returns:	0xHHHH<n><return> where H is a single hex digit.

READ TEMPERATURE

Description:	Displays temperature measured by sensor at location U3 in degrees Centigrade
Command:	`temp`
Returns:	`NN.N<return >` Where NN.N is the temperature in degrees centigrade. If less than 10, a leading zero is inserted.

RESET SOFTWARE

Description:	Issues a software reset to the processor. Used to simulate a power-on condition for testing. This command can take a second or so to execute.
Command:	`*RESET`
Returns:	"Power on" prompt.

RESET BOARD TO MANUFACTURED STATE

Description:	Clears the on-board EEPROM and issues a software reset (see above). This restores the board to factory manufactured state with the exception that the contents of the external flash memory (bitmap and macro storage) is not affected.
Command:	`*MFGRESET`
Returns:	"Power on" prompt.

DEBUG TOUCH

Description:	Used for Reach internal debugging, and serial output is subject to change at any time. When set, an "X" is written on the screen when a valid touch is detected and debug information is written to the serial port.
Command:	`*debug <0|1>`
Returns:	`[on|off]<return>`

DEBUG MACRO

Description:	Used to enable macro debug. When set, the commands in the macro are displayed as they are executed.
Command:	`*macdebug <0|1>`
Returns:	`[on|off]<return>`

MACRO NOTIFY

Description: This command sets the desired macro execution notification. This is used when a button or hotspot is assigned to a macro (see TOUCH MACRO ASSIGN). By default when an assigned macro executes there is no notification to the host other than the button response. For debugging and software interface verification purposes, the host can be notified when a touch-invoked macro is executed, when it finishes, or both.

Note that the notification is sent after the button press response.

Command: `*macnote <0|1|2|3>`

Arguments:
`0` – turn notification off.
`1` – send notification "`m<index><return>`" when macro starts.
`2` – send notification "`e<index><return>`" when macro ends.
`3` – send start and end notifications per 1 and 2 above.

Returns:
`off<return>` or
`start<return>` or
`end<return>` or
`both<return>`

POWER-ON MACRO

Description: Used to define a macro that is executed when the unit is first powered on. This can be used to set the desired baud rate if the default of 115,200 is too fast.

Note: the internally generated power-on copyright notice is displayed AFTER the power-on macro executes. This is done so the baud rate can be displayed. This can be disabled via the optional second parameter.

Note that the power-on copyright can also be suppressed by the splash screen option. If a splash screen is specified, the copyright notice is not displayed. The splash screen can be any bitmap, even a very small one that is the same color as the screen background.

Command: `*PONMAC <index> [<option>]`

Arguments:
(none) = display the current power-on macro index, or 0 for none.
`<index>` = 0 or 255 disables the power-on macro feature
`<index>` = 1 through 254 sets the power-on macro to the specified macro.
`<option>` = optional argument; 0 means display the power-on copyright, and 1 means do not display it.

Example: `*PONMAC 2`

BINARY NOTIFICATION MODE

Description: Used to set SLCD notification mode to binary or ASCII.

Due to parsing constraints, it is sometimes useful to have the SLCD provide notifications in fixed length binary format instead of variable length ASCII. This command provides the binary option.

Command: `*binr <0|1>`

Arguments:

0 Button / Hotspot / Macro notification is standard ASCII as specified in the button, and hotspot, and macro notify commands.

1 Button / Hotspot / Macro notification is in binary format as follows:

Standard (ASCII) notification	Binary notification
`x<index><return>`	`X<binary index><0x00>`
`r<index><return>`	`R<binary index><0x00>`
`s<index><state><return>`	`S<binary index><binary state><0x00>`
`m<index><return>`	`M<binary index><0x00>`
`e<index><return>`	`E<binary index><0x00>`
`<index> is 1-3 ASCII digits`	`<binary index> is a single byte`
	`<binary state> is a single byte`

Returns: `on<return>`

or

`off<return>`

SET DEMO MACRO

Description: Used to set the macro used for power-on demo. This macro will be executed if valid when the unit powers on and sees that the serial input is looped back. This is a simple way to include an optional self-running demo with evaluation kits.

Command: `*DEMOMAC <index>`

GET PANEL TYPE

Description: The unit's firmware is different depending on the panel and inverter it supports, even if the software version is the same. This command displays a human readable string that shows the panel definition used to create the firmware.

Command: `*panel`

SET CONTROL PORT

Description: Used to set the port used to control the unit. This is stored in EEPROM and will be used on power-up. Note that this switches between the RS232 and RS422 port but does not enable the rs485 (addressed) mode.

Command: `*com0main`

`*com1main`

SET PREVIOUS CONTROL PORT

Description: Used to revert to the previous port after a port autoswitch (three <return> characters on the inactive port.

Command: `*prevCons`

6. Fonts

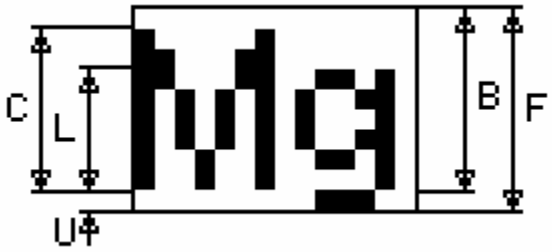
6.1. *Proportional Fonts*

Font 8 – ISO 8859-1 (Latin1 or Western European)

F: 08
B: 07
C: 07
L: 05
U: 01

Font 10 – ISO 8859-1 (Latin1 or Western European)

F: 10
B: 09
C: 08
L: 06
U: 01

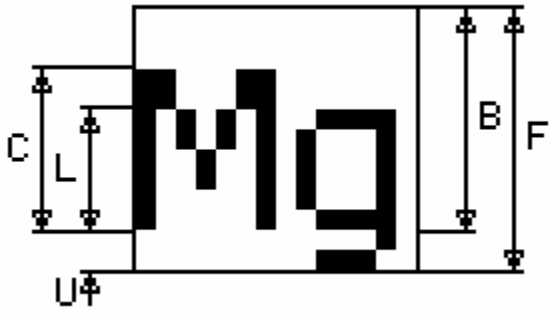


Font 10S – ISO 8859-1 (Latin1 or Western European)

F: 10
B: 08
C: 06
L: 04
U: 02

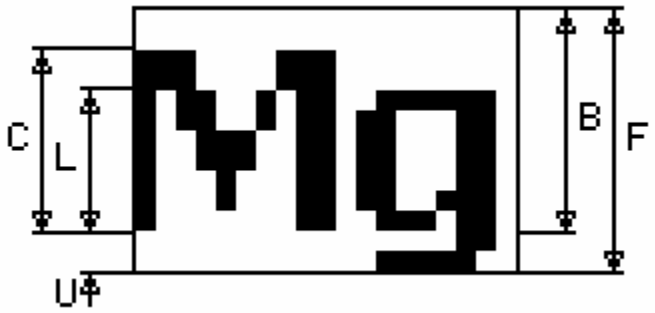
Font 13 – ISO 8859-1 (Latin1 or Western European)

F: 13
B: 11
C: 08
L: 06
U: 02



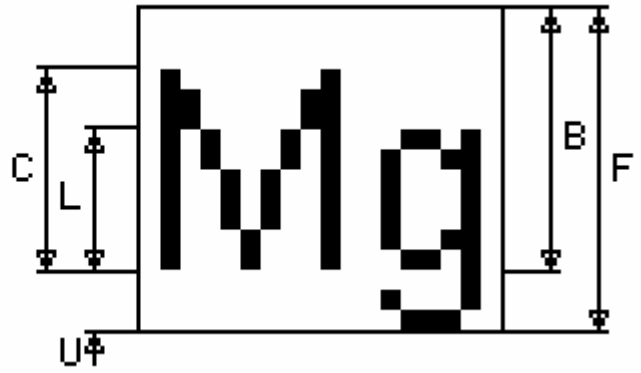
Font 13B – ISO 8859-1 (Latin1 or Western European)

F: 13
B: 11
C: 09
L: 07
U: 02



Font 16 – ISO 8859-1 (Latin1 or Western European)

F: 16
B: 13
C: 10
L: 07
U: 03

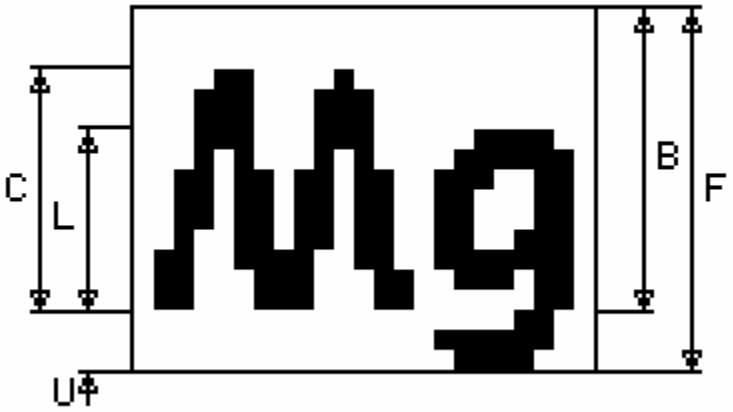


Font 16B – ISO 8859-1 (Latin1 or Western European)

F: 16
B: 13
C: 10
L: 07
U: 03

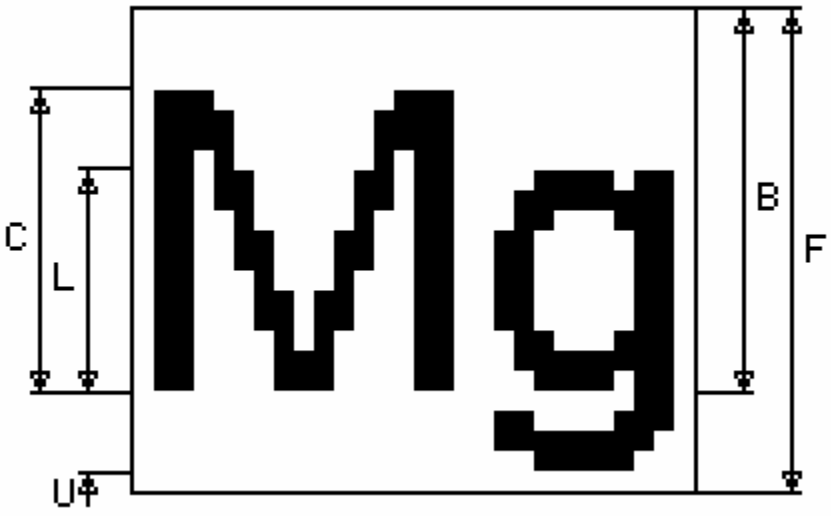
Font 18BC – ISO 8859-1 (Latin1 or Western European)

F: 18
B: 15
C: 12
L: 09
U: 03

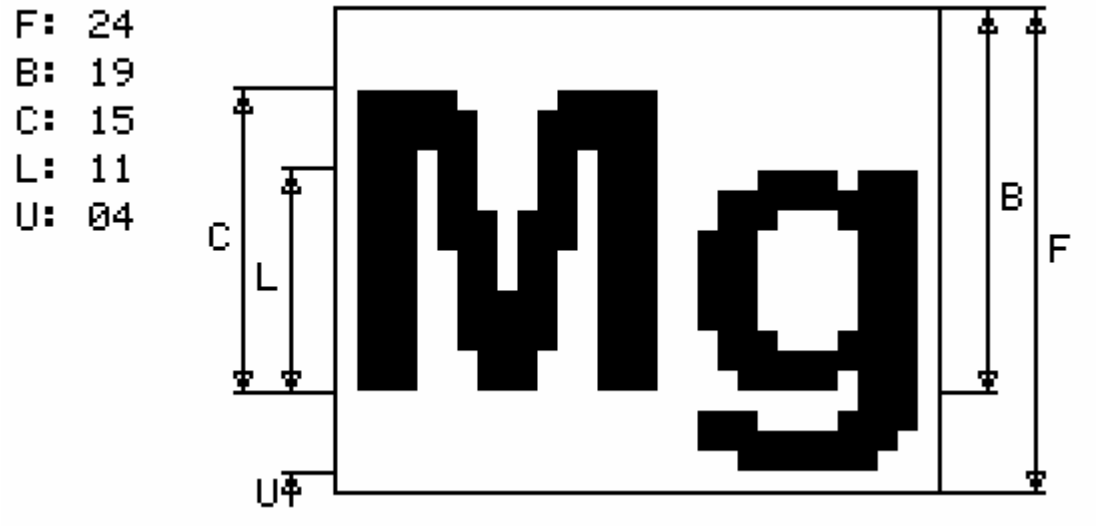


Font 24 – ISO 8859-1 (Latin1 or Western European)

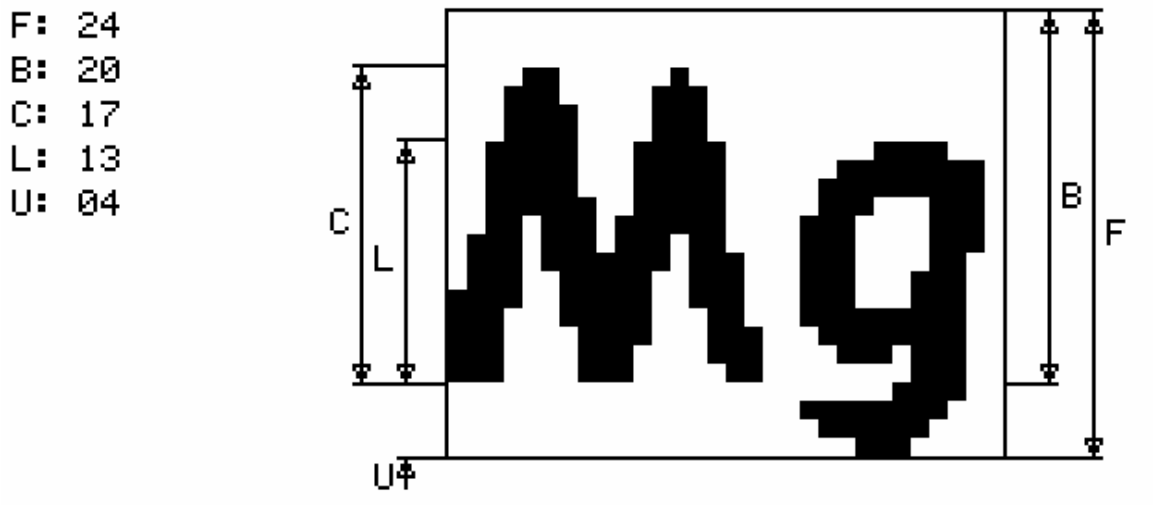
F: 24
B: 19
C: 15
L: 11
U: 04



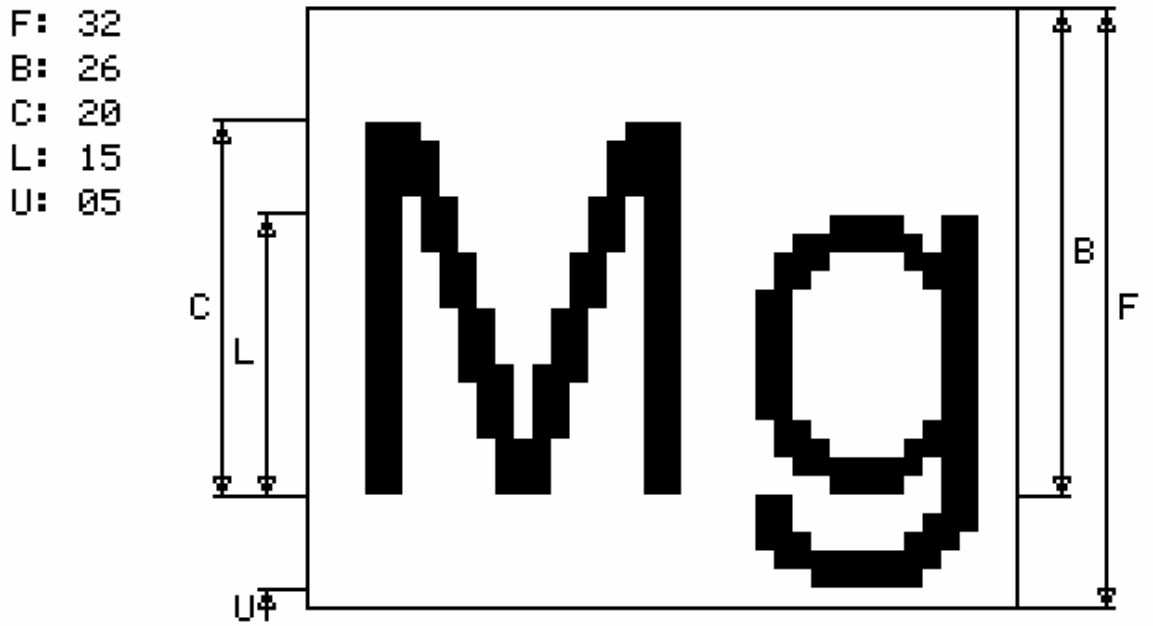
Font 24B – ISO 8859-1 (Latin1 or Western European)



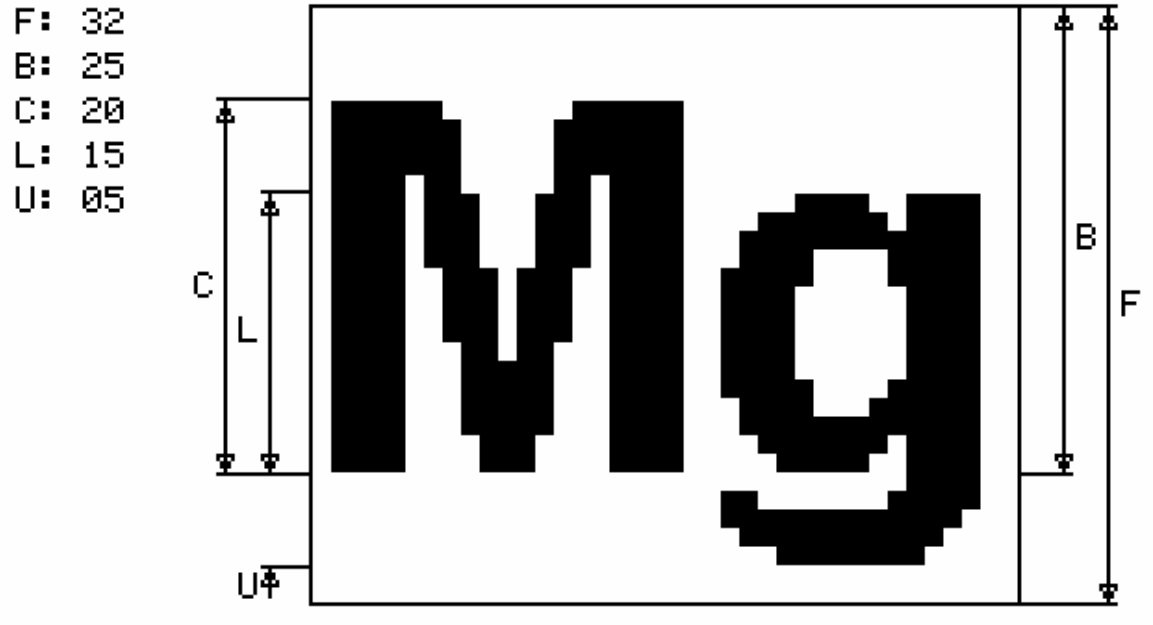
Font 24BC – ISO 8859-1 (Latin1 or Western European)



Font 32 – ISO 8859-1 (Latin1 or Western European)



Font 32B – ISO 8859-1 (Latin1 or Western European)



6.2. *Monospaced Fonts*

Font 4x6 – ASCII Only

F: 06
B: 05
C: 05
L: 04
U: 01

Font 6x8 – ISO 8859-1 (Latin1 or Western European) *EXTENDED*

F: 08
B: 07
C: 07
L: 05
U: 01

Font 6x9 – ISO 8859-1 (Latin1 or Western European) *EXTENDED*

F: 09
B: 07
C: 07
L: 05
U: 01

Font 8x8 – ISO 8859-1 (Latin1 or Western European) Extended

F: 08
B: 07
C: 07
L: 05
U: 01

Font 8x9 – ISO 8859-1 (Latin1 or Western European) *EXTENDED*

F: 09
B: 07
C: 07
L: 05
U: 01

Font 8x10 – ASCII Only

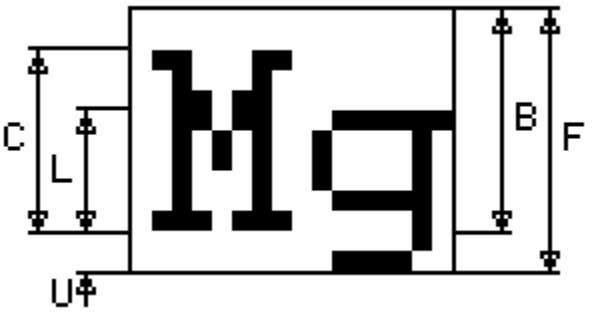
F: 10
B: 09
C: 09
L: 07
U: 01

Font 8x12 – ASCII Only

F: 12
B: 10
C: 09
L: 06
U: 02

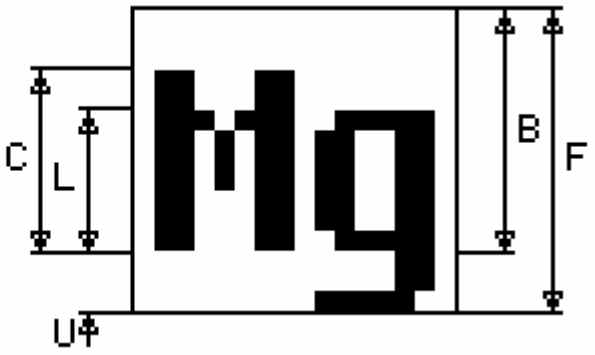
Font 8x13 – ASCII Only

F: 13
B: 11
C: 09
L: 06
U: 02



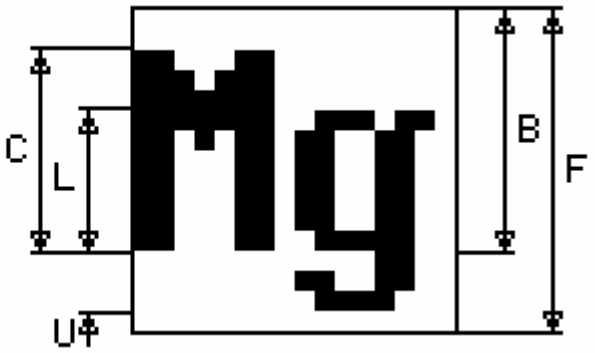
Font 8x15B – ASCII Only

F: 15
B: 12
C: 09
L: 07
U: 03



Font 8x16 – ISO 8859-1 (Latin1 or Western European) *EXTENDED*

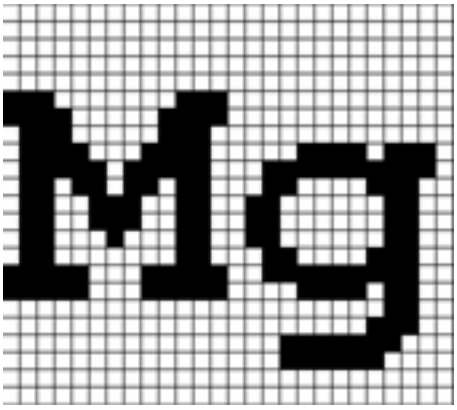
F: 16
B: 12
C: 10
L: 07
U: 03



Font 8x16L

Same as 8x16 except the numbers 0-9 are "light"

Font 14x24 – ISO 8859-1



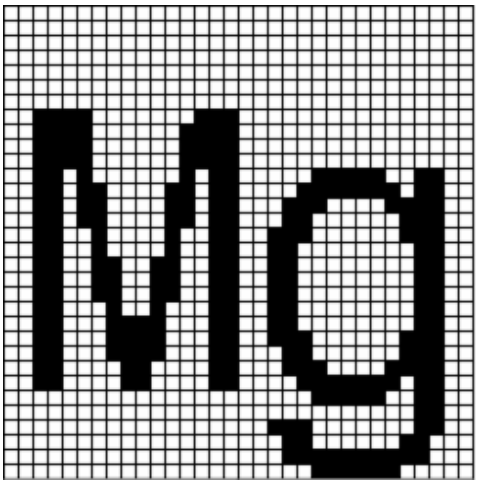
Font 16x32 – ISO 8859-1

This is the font 8x16 doubled in both directions:

```
F: 32
B: 24
C: 20
L: 14
U: 06
```

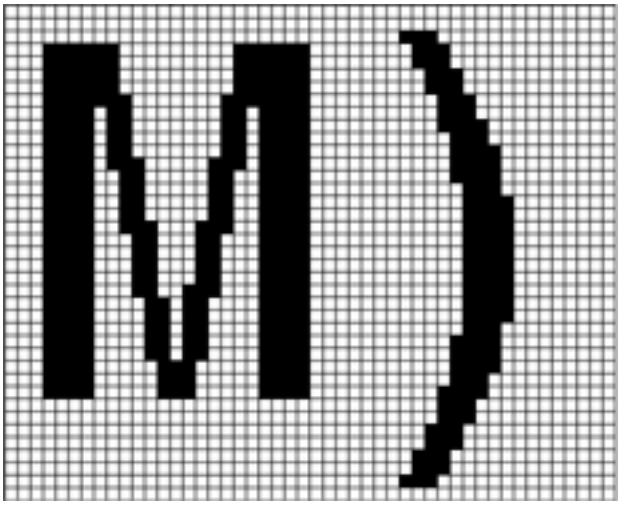
Font 16x32i – ISO 8859-1

This is an improved version of the 8x16 above.



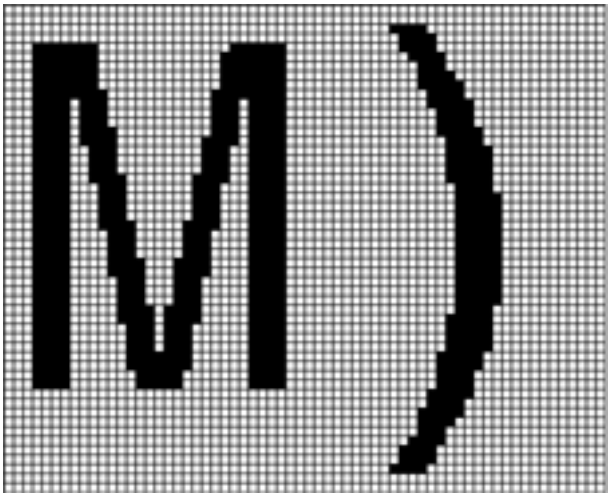
Font 24x48 – Numbers, Capital letters, Symbols

Note: The actual character size is 24x39 pixels; the font is 48 point.



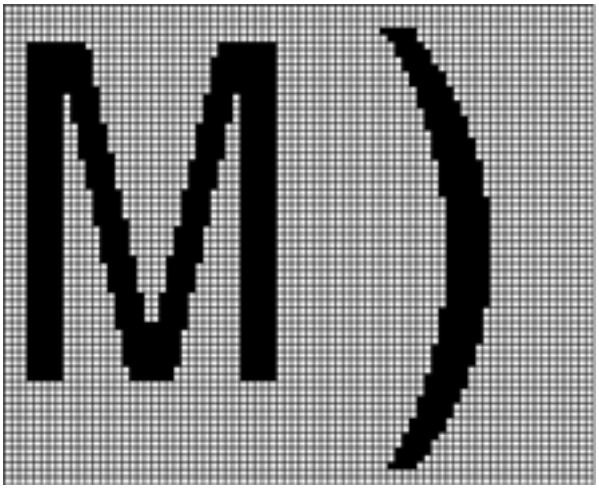
Font 32x64 – Numbers, Capital letters, Symbols

Note: The actual character size is 32x52 pixels; the font is 64 point.



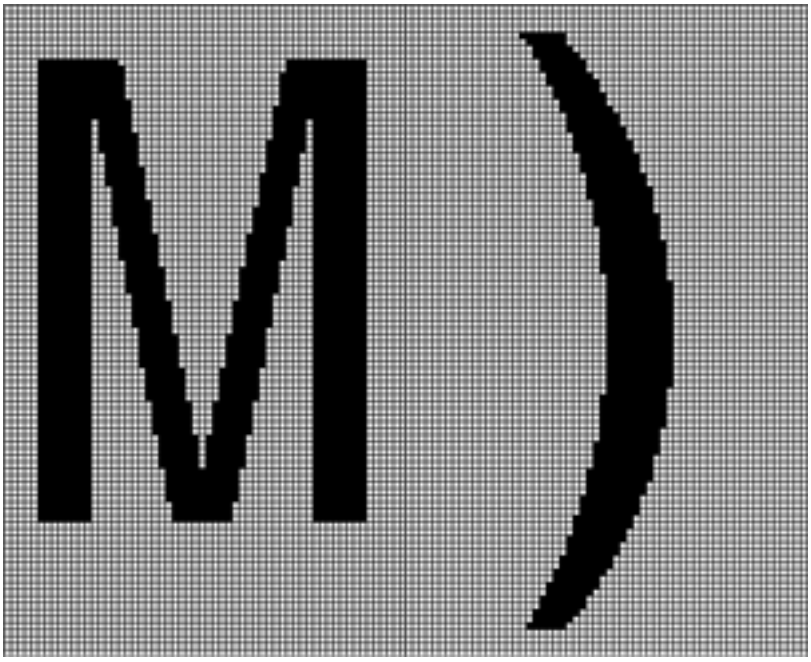
Font 40x80 – Numbers, Capital letters, Symbols

Note: The actual character size is 40x65 pixels; the font is 80 point.



Font 60x120 – Numbers, Capital letters, Symbols

Note: The actual character size is 60x97 pixels; the font is 120 point.



6.3. *Character Set - ISO 8859-1*

The ISO 8859-1 character set used by most fonts is as follows. Note that the ASCII character set is the same as the ISO up to Code 127. The ISO set does not define characters 0-31, or 127-159. The extended ISO set includes characters 144-149 per the table shown.

Char	Code	Name	Description
	32	-	Normal space
!	33	-	Exclamation
"	34	quot	Double quote
#	35	-	Hash
$	36	-	Dollar
%	37	-	Percent
&	38	amp	Ampersand
'	39	-	Apostrophe
(	40	-	Open bracket
)	41	-	Close bracket
*	42	-	Asterisk
+	43	-	Plus sign
,	44	-	Comma
-	45	-	Minus sign
.	46	-	Period
/	47	-	Forward slash

Char	Code	Name	Description
0	48	-	Digit 0
1	49	-	Digit 1
2	50	-	Digit 2
3	51	-	Digit 3
4	52	-	Digit 4
5	53	-	Digit 5
6	54	-	Digit 6
7	55	-	Digit 7
8	56	-	Digit 8
9	57	-	Digit 9
:	58	-	Colon
;	59	-	Semicolon
<	60	lt	Less than
=	61	-	Equals
>	62	gt	Greater than
?	63	-	Question mark

Char	Code	Name	Description
@	64	-	At sign
A	65	-	A
B	66	-	B
C	67	-	C
D	68	-	D
E	69	-	E
F	70	-	F
G	71	-	G
H	72	-	H
I	73	-	I
J	74	-	J
K	75	-	K
L	76	-	L
M	77	-	M
N	78	-	N
O	79	-	O

Char	Code	Name	Description
P	80	-	P
Q	81	-	Q
R	82	-	R
S	83	-	S
T	84	-	T
U	85	-	U
V	86	-	V
W	87	-	W
X	88	-	X
Y	89	-	Y
Z	90	-	Z
[	91	-	Open square bracket
\	92	-	Backslash
]	93	-	Close square bracket
^	94	-	Caret
_	95	-	Underscore

Char	Code	Name	Description
`	96	-	Grave accent
a	97	-	a
b	98	-	b
c	99	-	c
d	100	-	d
e	101	-	e
f	102	-	f
g	103	-	g
h	104	-	h
i	105	-	i
j	106	-	j
k	107	-	k
l	108	-	l
m	109	-	m
n	110	-	n
o	111	-	o

Char	Code	Name	Description
p	112	-	p
q	113	-	q
r	114	-	r
s	115	-	s
t	116	-	t
u	117	-	u
v	118	-	v
w	119	-	w
x	120	-	x
y	121	-	y
z	122	-	z
{	123	-	Left brace
|	124	-	Vertical bar
}	125	-	Right brace
~	126	-	Tilde
	127	-	(Unused)

Char	Code	Name	Description
	160	nbsp	Non-breaking space
¡	161	iexcl	Inverted exclamation
¢	162	cent	Cent sign
£	163	pound	Pound sign
¤	164	curren	Currency sign
¥	165	yen	Yen sign
¦	166	brvbar	Broken bar
§	167	sect	Section sign
¨	168	uml	Umlaut or diaeresis
©	169	copy	Copyright sign
ª	170	ordf	Feminine ordinal
«	171	laquo	Left angle quotes
¬	172	not	Logical not sign
-	173	shy	Soft hyphen
®	174	reg	Registered trademark
¯	175	macr	Spacing macron

Char	Code	Name	Description
°	176	deg	Degree sign
±	177	plusmn	Plus-minus sign
²	178	sup2	Superscript 2
³	179	sup3	Superscript 3
´	180	acute	Spacing acute
µ	181	micro	Micro sign
¶	182	para	Paragraph sign
·	183	middot	Middle dot
¸	184	cedil	Spacing cedilla
¹	185	sup1	Superscript 1
º	186	ordm	Masculine ordinal
»	187	raquo	Right angle quotes
¼	188	frac14	One quarter
½	189	frac12	One half
¾	190	frac34	Three quarters
¿	191	iquest	Inverted question mark

Char	Code	Name	Description
À	192	Agrave	A grave
Á	193	Aacute	A acute
Â	194	Acirc	A circumflex
Ã	195	Atilde	A tilde
Ä	196	Auml	A umlaut
Å	197	Aring	A ring
Æ	198	AElig	AE ligature
Ç	199	Ccedil	C cedilla
È	200	Egrave	E grave
É	201	Eacute	E acute
Ê	202	Ecirc	E circumflex
Ë	203	Euml	E umlaut
Ì	204	Igrave	I grave
Í	205	Iacute	I acute
Î	206	Icirc	I circumflex
Ï	207	Iuml	I umlaut

Char	Code	Name	Description
Ð	208	ETH	ETH
Ñ	209	Ntilde	N tilde
Ò	210	Ograve	O grave
Ó	211	Oacute	O acute
Ô	212	Ocirc	O circumflex
Õ	213	Otilde	O tilde
Ö	214	Ouml	O umlaut
×	215	times	Multiplication sign
Ø	216	Oslash	O slash
Ù	217	Ugrave	U grave
Ú	218	Uacute	U acute
Û	219	Ucirc	U circumflex
Ü	220	Uuml	U umlaut
Ý	221	Yacute	Y acute
Þ	222	THORN	THORN
ß	223	szlig	sharp s

Char	Code	Name	Description
à	224	agrave	a grave
á	225	aacute	a acute
â	226	acirc	a circumflex
ã	227	atilde	a tilde
ä	228	auml	a umlaut
å	229	aring	a ring
æ	230	aelig	ae ligature
ç	231	ccedil	c cedilla
è	232	egrave	e grave
é	233	eacute	e acute
ê	234	ecirc	e circumflex
ë	235	euml	e umlaut
ì	236	igrave	i grave
í	237	iacute	i acute
î	238	icirc	i circumflex
ï	239	iuml	i umlaut

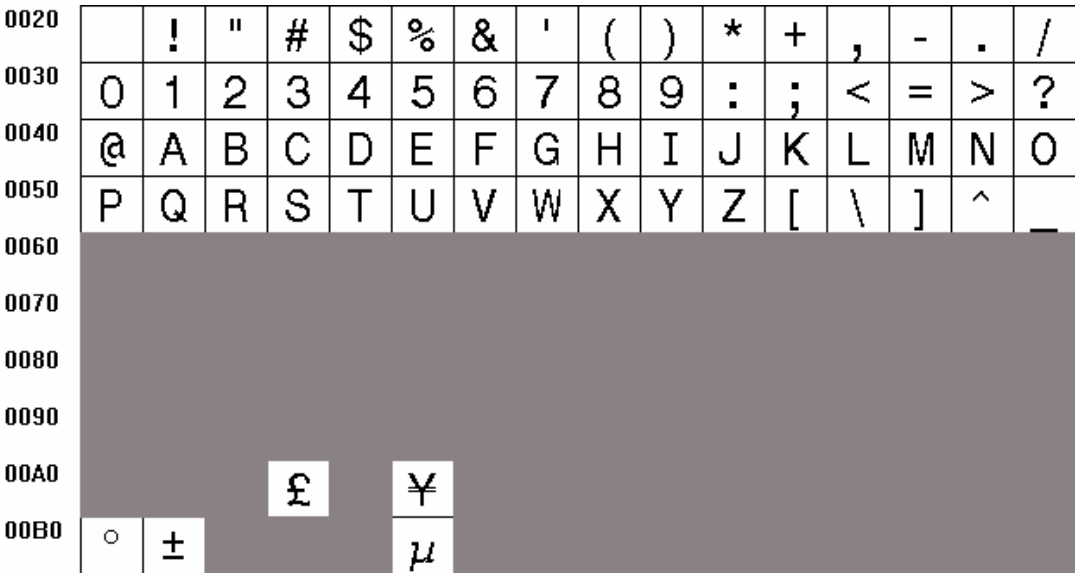
Char	Code	Name	Description
ð	240	eth	eth
ñ	241	ntilde	n tilde
ò	242	ograve	o grave
ó	243	oacute	o acute
ô	244	ocirc	o circumflex
õ	245	otilde	o tilde
ö	246	ouml	o umlaut
÷	247	divide	division sign
ø	248	oslash	o slash
ù	249	ugrave	u grave
ú	250	uacute	u acute
û	251	ucirc	u circumflex
ü	252	uuml	u umlaut
ý	253	yacute	y acute
þ	254	thorn	thorn
ÿ	255	yuml	y umlaut

***EXTENDED* ISO characters:**

←	144	left arrow
→	145	right arrow
↑	146	up arrow
↓	147	down arrow
↵	148	enter symbol
✓	149	checkmark

6.4. *Character Set - Numbers, Capital letters, Symbols*

The large monospaced fonts provide a reduced character set of the ISO 8859-1 as follows:



Appendix A - Models and Ordering Information

The unit is available with different display panels as follows. Each panel includes a factory mounted touchscreen.

Model	**Ordering Part Number**
Terminal with Hitachi TFT panel TX14D11VM1CAA	42-0080-01
Terminal with Hitachi CSTN panel SX14Q001-ZZA	42-0076-01
Terminal with Kyocera TFT panel TCG057QV1AD-G00	42-0077-01

Optional accessories are also available:

Accessory	**Ordering Part Number**
Power supply: Universal input 12VDC 1A output supply with attached two position Phoneix Contact connector	30-0005-02
RS232 PC Serial cable: DB9 female to three position Phoneix Contact connector. Length 36".	23-0085-36A

Appendix B - BMPload program

B.1 *Overview*

The SLCD contains 512Kbytes of flash memory used for storing bitmaps and macros. Stored bitmaps are displayed on the screen using the "xi" command (See **DISPLAY DOWNLOADED BITMAP IMAGE** Command). The BMPload.exe program is used to transfer bitmaps and a macro file from the PC to the SLCD flash memory. Once downloaded, the images are non-volatile; that is they are permanently stored even if power is off.

The download process clears the entire flash memory.

B.2 *Bitmap Format*

The SLCD requires bitmaps to be less than or equal to 320 by 240 pixels, 8 bit indexed color. This is also known as 256 color or palletized color mode. This applies to both black and white and color displays. Black and white displays can only show 16 gray levels. Bitmaps can be created with programs such as the Windows PAINT program or by Adobe PhotoShop. The PhotoShop palette file ps8666.act contains the palette used on the SLCD. Use this to palletize your bitmaps to use less storage and display faster. See Appendix H for more details on how to generate bitmaps.

B.3 *Program Operation*

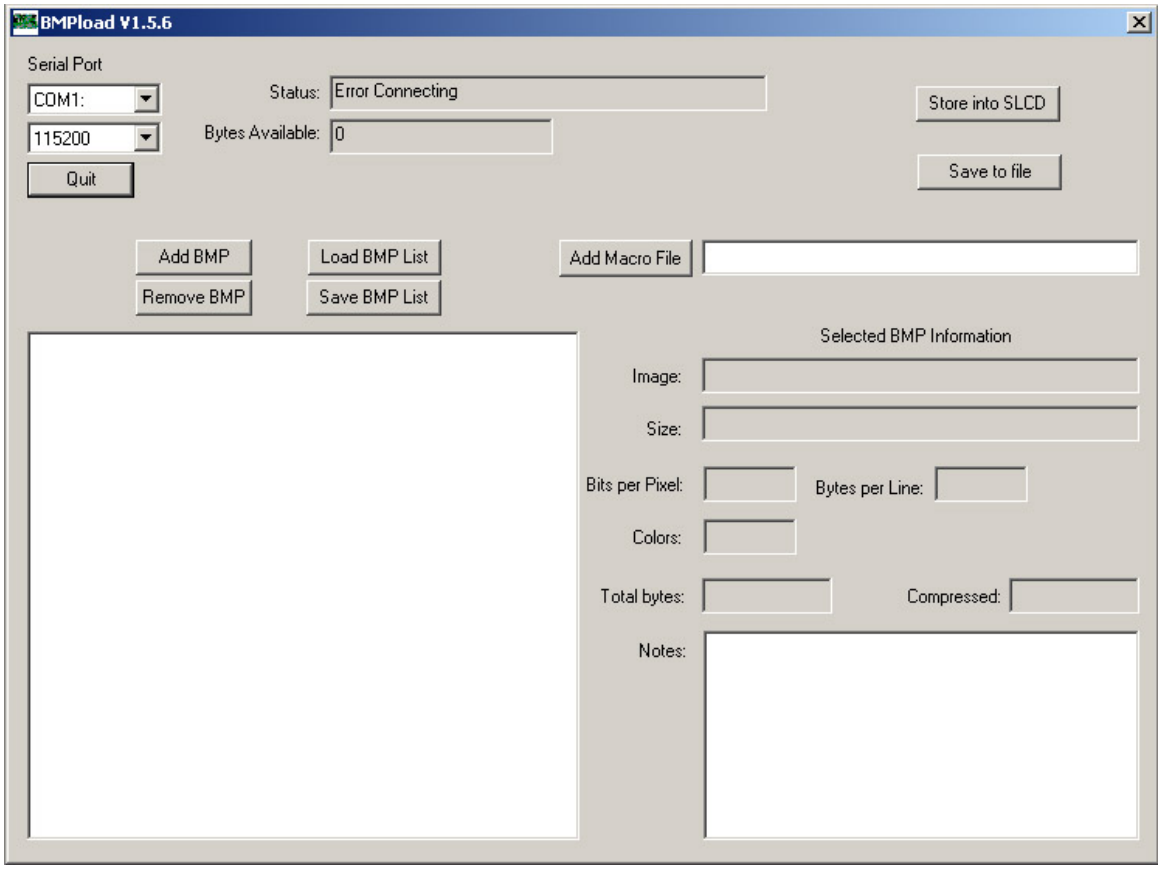
BMPload runs under Windows 98 through XP. The computer running BMPload must have a serial port connected to the SLCD board. The serial port must not be in use by another program. When it is first run, you may see the following:

This means that the program failed to open the default COM1 port at its default baud rate. This may be due to another program such as HyperTerminal being open and connected to COM1. Shut down or disconnect any other serial programs, and click OK.

You may see the following message instead:

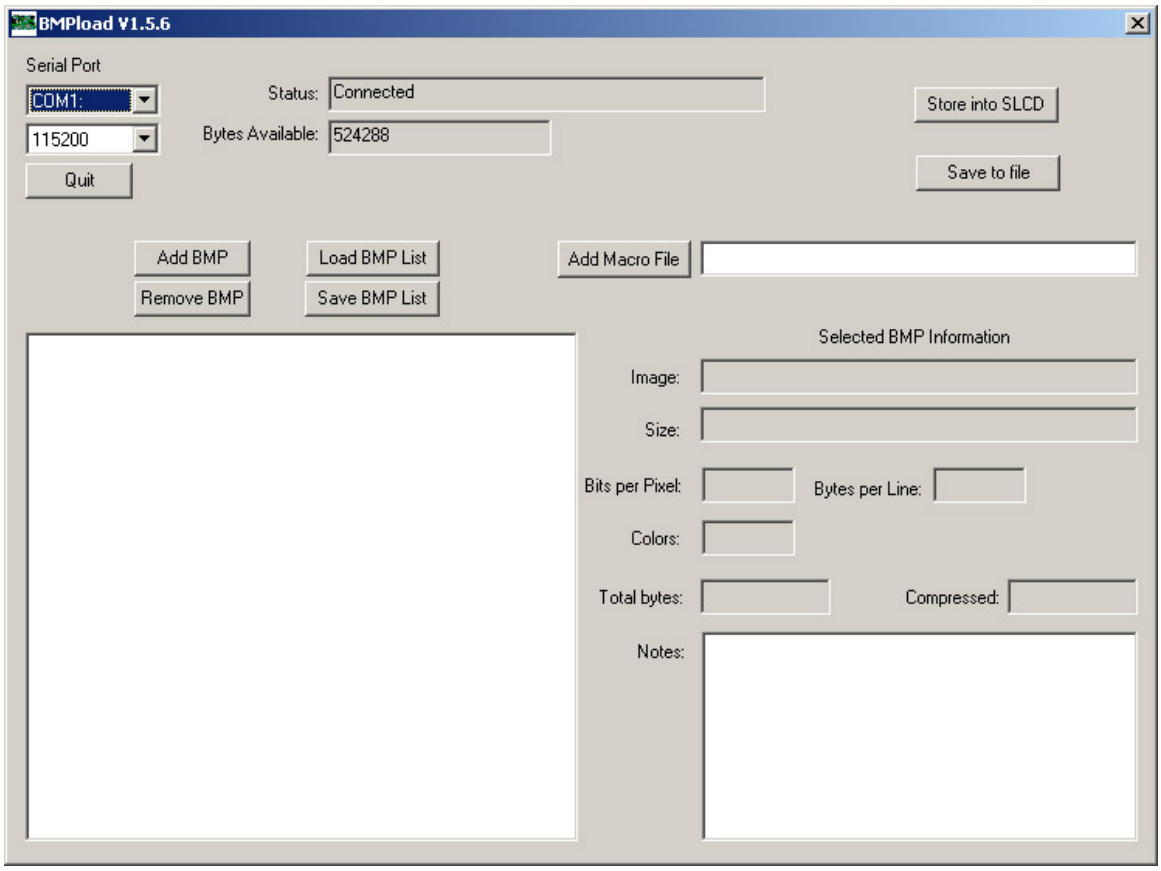
This means that the serial connection between the PC and the SLCD is not working. Check the cables.

Once open, the program looks like this if the SLCD is not connected or connected to other than COM1:



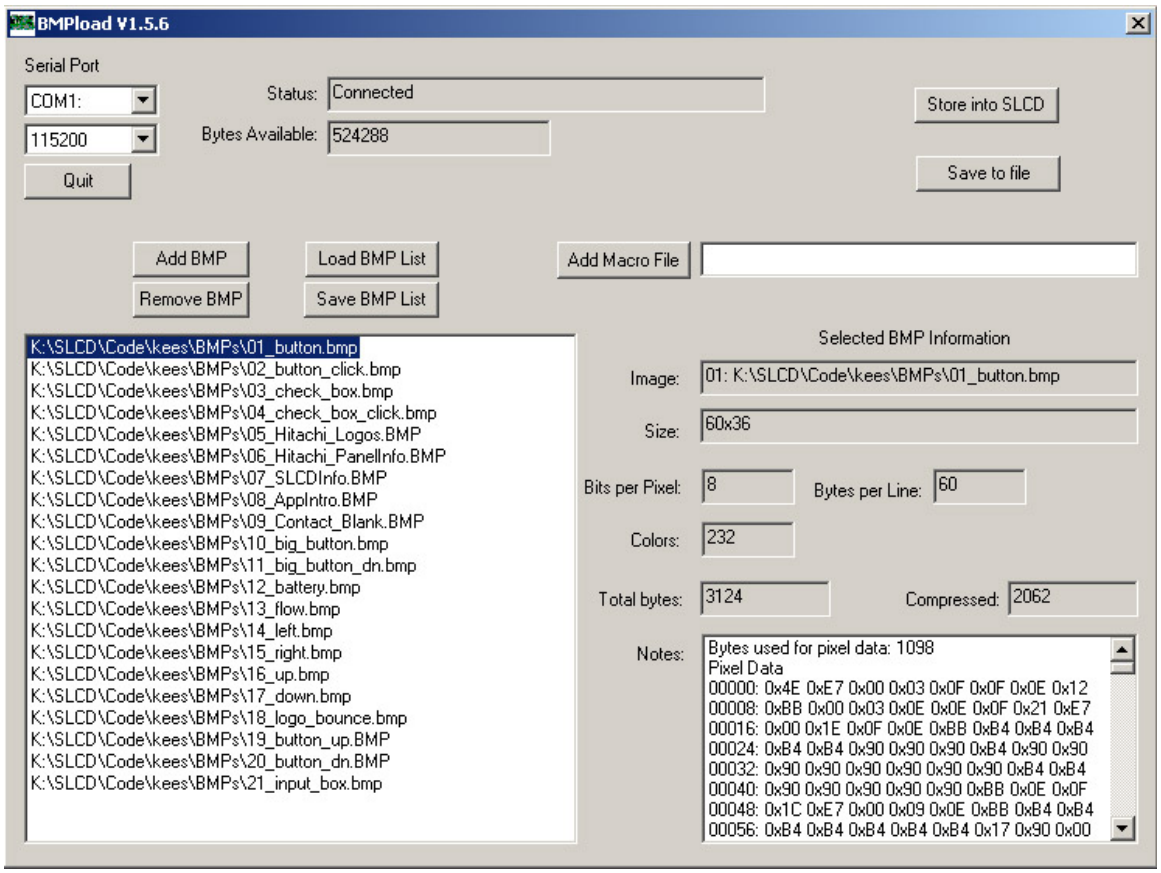
Change the serial port from COM1 to the appropriate port, or back to COM1 to try connecting again.

The program looks like this if the SLCD is correctly connected via COM1. You can also connect via COM2 through COM4 by using the drop-down box.



You can now use the "Add BMP" button to add BMP files to the list. Note that the order is important because you use the index number in the DISPLAY DOWNLOADED BITMAP IMAGE command. The best way to keep this clear is to start the bmp file name with its index number, for example "01_first_bitmap.bmp"

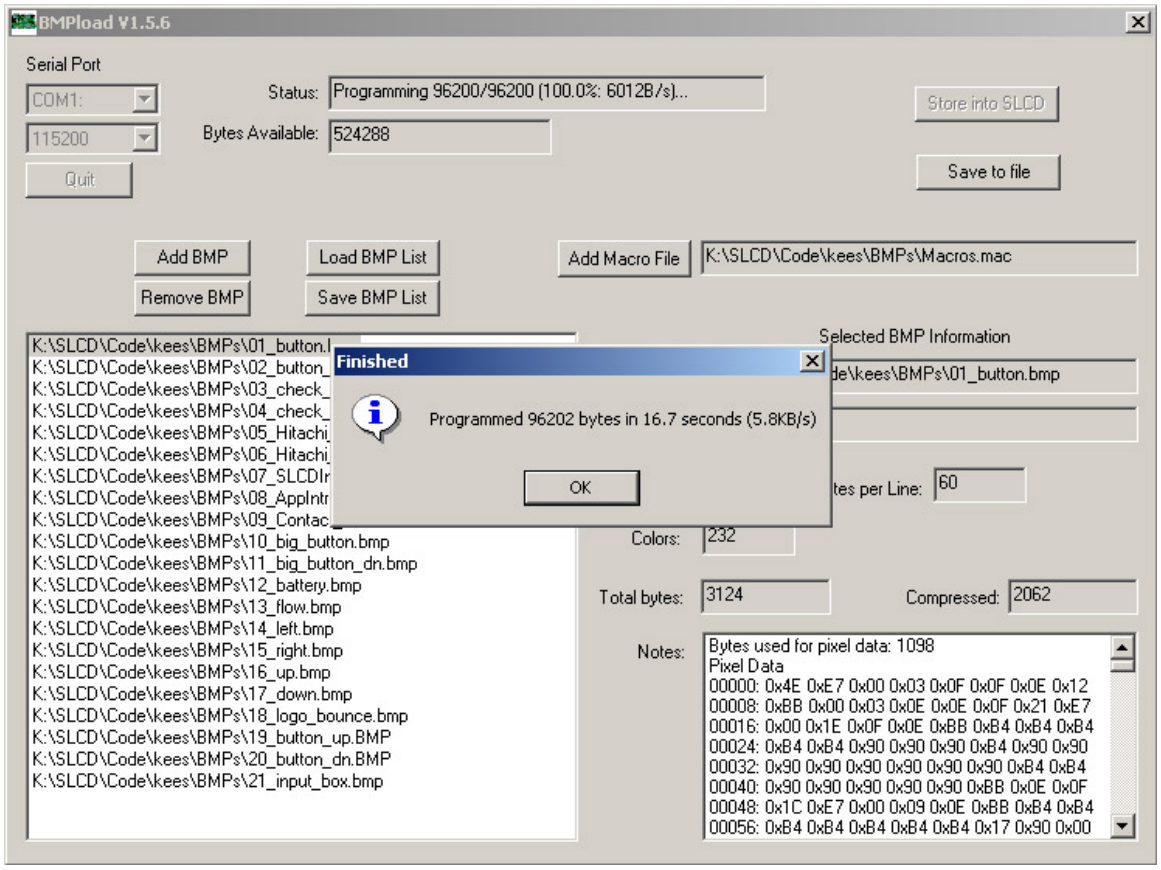
Once added, each BMP can be highlighted and detailed information will display on the right hand side. Bitmaps are compressed for storage using the RLE algorithm.



Use the "Add Macro File" button to add a macro file to the flash memory as well as the bitmaps.

Once you have added the bitmaps, use the "Store BMPs" button to save them to the SLCD.

When the program has finished loading, you will see the following display.



You can save and load lists of bitmaps for convenience. These are text files containing the list of bitmaps. Use the "Save BMP list" button and edit the saved file to see the format. You can edit the file with any text editor and load the list using the "Load BMP list" command. Note that by naming the bitmaps with a numeric prefix in the order they are loaded, it is easy to remember which bitmap is assigned to which bitmap number. This is needed for the "xi" command that uses bitmap numbers not file names.

B.4 *BMPLoad speed issues*

The BMPload program will work with any type of serial port. However, some USB-to-serial converters have high overhead for the small transaction sizes used by the BMPload program. If you are seeing slow programming times with a USB converter, try using a standard serial port.

Appendix C – Macro commands and file format

C.1 *Introduction and limitations*

Macros have two main purposes.

1) They allow a series of commands to be invoked by a single command. This can speed up the display by reducing communication overhead. It also reduces the space needed to store commands on the host processor

2) They can be linked to buttons so that by pushing a button, a macro can generate a new screen. This is useful to keep the overhead on the processor low and provide fast response for users.

Macros can have parameters associated with them. This allows a general purpose macro to be used in different ways. For example, a macro could create a numeric keypad and the parameters would specify where to draw the keypad on the screen. This reduces hard coding of graphical elements and promotes reuse between screens and products.

There are version-dependent limits on the macro commands and their arguments. For firmware version 2.3.0 and above, those limits are:

MAXIMUM CALL DEPTH = 4

A macro can call another macro, but only to a depth of 4.

MAXIMUM ARGUMENTS PER MACRO = 4

MAXIMUM CHARACTERS PER ARGUMENT = 8

MAXIMUM TOTAL STORED ARGUMENTS = 50 (stored via the TOUCH MACRO ASSIGN WITH ARGUMENTS command)

C.2 *Macro File Format*

The macro file is an ASCII text file and can be generated by Windows applications such as Notepad. The file format is designed so that the same macro definition file can be used a) to load the macros into the SLCD flash memory, and b) as a 'C' include file in the user's microcontroller program. This way there is only one file to avoid confusion with macro index numbers.

The format for each macro is as follows:

#define <text_name> <number>

(one or more command lines)

.

.

#end

The <text_name> is an identifier that follows 'C' language conventions, and is included for reference if the macro file is included in a C program. It has no other use.

The <number> argument must be 1 for the first macro, 2 for the second, and so on. The macros must be listed in increasing contiguous index order.

Comments are ignored. Comments are lines starting with the '/' forward slash symbol. All lines outside of a "#define…#end pair are treated as comments. By using 'C' style comments in a creative way, only the #define lines are seen by the C program.

C.3 *Macro Parameters (Arguments)*

Macros can be parameterized by using the special escape sequences `` `0`, `1`, `2`, `` and `` `3` `` in the command lines. These are replaced at execution time by the arguments supplied by the command that invoked the macro.

Parameterized macro example:

```
#define example 1
t "`0`" `1` `2`
#end
```

The following command uses this macro to display the text "Hello" at location x=10, y=20:

```
>m 1 Hello 10 20
```

C.4 *Special macro arguments and commands*

Memory commands

Memory commands were added to implement the keyboard in the demo macro that comes installed with the SLCD kits. These allow a character string to be saved and manipulated. See Section E.6 for examples. The character string is accessed as a special macro parameter.

The commands are::

`mpush "<string>"`

This appends the string argument to the memory variable. The maximum stored string length is 80 characters

Example: `mpush "0"`

`mpop <number>`

This removes the <number> of characters specified from the end of the memory variable.

Example: `mpop 1`

Internal Arguments

The macro system recognizes other symbols in the parameter esacpe format – enclosed in back tick marks. These are as follows.

Memory variable

`` `M` ``

This is replaced by the string stored by the mpush command. See the demo macro in Section E.6 for examples.

Random number

`` `R<lo>:<hi>` ``

This is replaced by a random number in the range <lo> to <hi>; see the demo macro in Section E.6 for examples.

Repeat command

A special command allows a macro to repeat execution. The command is:

```
:repeat
```

When the macro processor reads this line, the macro will begin execution again at the first line of the macro. NOTE: An escape character (hex 1B) followed by a <return> received from the serial port will halt a looping macro.

C.5 *Changing the power-on baud rate*

The following is an example of how to set the power-on baud rate. Create and load the following macro file:

```
#define pon_mac 1
/* (start comment out contents)...
// set baud rate
baud 9600
#end */
```

Now, connect to the SLCD and run the command:

```
*PONMAC 1<return>
```

Now cycle power to the SLCD, and the initial baud rate will be 9600 baud.

C.6 *Macro Example – (factory loaded into SLCD flash)*

The following file is factory loaded into the SLCD external flash. It is used to implement the self-running demo that starts if a loopback is detected on the SLCD serial port on power-on.

The demo macro is number 1 which runs through a series of screens and shows how to use parameterized macros and buttons that invoke macros with parameters.

Macro 6 is a simpler example that displays a keypad and a single digit when any of the keys are pushed.

```
//
// Reach SLCD macro demo file
// 2-1-2005
//
//---------------------------------------------------------------------
//  This macro file assumes the following bitmaps are loaded (in order):
//
// 01_button.bmp
// 02_button_click.bmp
// 03_check_box.bmp
// 04_check_box_click.bmp
// 05_Hitachi_Logos.BMP
// 06_Hitachi_PanelInfo.BMP
// 07_SLCDInfo.BMP
// 08_Contact_Blank.BMP
// 09_logo_bounce.bmp
// 10_button_up.BMP
// 11_button_dn.BMP
// 12_input_box.bmp
//
//---------------------------------------------------------------------

/* comments can be used in either 'C' style or C++ */
// blank lines are allowed
// We comment out the contents of the macro so this complete file
// can be used as a C include file

//---------------------------------------------------------------------

//---------------------------------------------------------------------
// MACRO #1
// This is the macro that invokes the self-running demo.
// It is the default value for the *DEMOMAC command, so it starts
// if the SLCD is powered on with its serial port looped back to itself.
//---------------------------------------------------------------------
#define power_on_loopback_demo 1 /*
m29  // check for touch calibration
m10
m12
#end */


//---------------------------------------------------------------------
// MACROS #2 - #6
// These macros implement a number pad at relative (0,0)
// It assumes button BMP 19, 20 are loaded
//---------------------------------------------------------------------
```

```
#define number_pad 2 /*
// bitmap that holds displayed number
xi 12 0 0
// set font 24 for buttons
f 24
// define buttons
bd 1 0 32 1 "1" 9 5 10 11
bd 2 32 32 1 "2" 9 5 10 11
bd 3 64 32 1 "3" 9 5 10 11
bd 4 0 64 1 "4" 9 5 10 11
bd 5 32 64 1 "5" 9 5 10 11
bd 6 64 64 1 "6" 9 5 10 11
bd 7 0 96 1 "7" 9 5 10 11
bd 8 32 96 1 "8" 9 5 10 11
bd 9 64 96 1 "9" 9 5 10 11
bd 10 0 128 1 "*" 9 5 10 11
bd 0 32 128 1 "0" 9 5 10 11
bd 11 64 128 1 "#" 9 5 10 11
// tell user we're done
beep 10
/* a comment here only works if C compiler handles nested comments */
#end
   ...(end comment out contents) */

//---------------------------------------------------------------------
// Macro to set font and color for input box display. Foregorund color
// is XOR of background color to match the buttons
//---------------------------------------------------------------------
#define grey24 3 /*
S 333 CCC
f 24
#end  */

//---------------------------------------------------------------------
// These macros write text to the screen
//---------------------------------------------------------------------
#define button_text 4 /*
m2
t "`0`" 10 4
#end  */


//---------------------------------------------------------------------
// This macro enables the buttons specified in number_pad macro
// to write text to the screen inside a bitmap
//---------------------------------------------------------------------
#define attach_buttons 5 /*
xa 0 p 3 0
xa 1 p 3 1
xa 2 p 3 2
xa 3 p 3 3
xa 4 p 3 4
xa 5 p 3 5
xa 6 p 3 6
xa 7 p 3 7
xa 8 p 3 8
xa 9 p 3 9
#end  */

//---------------------------------------------------------------------
// This macro creates a keypad and displays the key
// using previous macros
//---------------------------------------------------------------------
```

```
#define keypad_demo 6 /*
s 0 1
m1
m4
#end  */

#define rand_draw 7 /*
p `R1:5`
m8
tr `R0:319` `R0:189` `R0:319` `R0:189` `R0:319` `R0:189`
m8
c `R0:319` `R0:159` `R10:40`
:repeat
#end */

#define rand_color 8 /*
s `R0:16` 1
#end */

#define demo_end 9 /*
m10
#end */

#define clear 10 /*
s 0 1
z
#end */

#define next_button 11 /*
o 0 0
f 13B
bd 1 257 202 1 "Next" 16 11 1 2
xm 1 `0`
#end */

#define splash_logos 12 /*
xi 5 0 0
m11 13
#end */

#define splash_panel 13 /*
xi 6 0 0
m11 14
#end */

#define splash_slcd 14 /*
xi 7 0 0
m11 15
#end */

#define splash_fonts 15 /*
z
f 24BC
t "On-board Proportional Fonts" 10 5
sc 0 0
o 20 40
f 8
t "8 point font: quick brown fox 0123"
f 16B
t "\n"
f 10
t "10 point: quick brown fox 0123"
f 16B
```

```
t "\n"
f 13
t "13 point: quick brown fox 0123"
f 16B
t "\n"
f 13B
t "13 point bold: quick brown fox 0123"
f 16B
t "\n"
f 16
t "16 point: quick brown fox 0123"
f 16B
t "\n"
f 16B
t "16 point bold: quick brown fox 0123"
f 18BC
t "\n18 point bold comic: quick fox 0123"
t "\n"
f 24
t "24 point: quick fox 0123"
f 24B
t "\n24 point bold: quick 01\n"
f 32
t "32 point & "
f 32B
t "32 bold"
m11 31
#end */

#define splash_keyboard 16 /*
z
// clear string memory
mpop -1
f 24BC
t "Easy to use buttons" 54 0
f 13B
// set up stateful button with macro callback
m17 "Off"
bd 2  50 24 2 "" "" 0 0 0 0 3 4
xa 2 p 17 "On  "
xa 2 r 17 "Off"
// setup demo instant button
bd 3 200 20 1 "Hold" 16 11 1 2
// adjust origin for keyboard
o 0 72
// keyboard row one
m18  10   0   0 1
m18  11  32   0 2
m18  12  64   0 3
m18  13  96   0 4
m18  14 128   0 5
m18  15 160   0 6
m18  16 192   0 7
m18  17 224   0 8
m18  18 256   0 9
m18  19 288   0 0
// keyboard row two
m18  20   0  32 Q
m18  21  32  32 W
m18  22  64  32 E
m18  23  96  32 R
m18  24 128  32 T
m18  25 160  32 Y
```

```
m18  26 192  32 U
m18  27 224  32 I
m18  28 256  32 O
m18  29 288  32 P
// keyboard row three
m18  30   0  64 A
m18  31  32  64 S
m18  32  64  64 D
m18  33  96  64 F
m18  34 128  64 G
m18  35 160  64 H
m18  36 192  64 J
m18  37 224  64 K
m18  38 256  64 L
m18  39 288  64 " "
// keyboard row four
m18  40   0  96 Z
m18  41  32  96 X
m18  42  64  96 C
m18  43  96  96 V
m18  44 128  96 B
m18  45 160  96 N
m18  46 192  96 M
m18  47 224  96 ,
m18  48 256  96 -
// special erase key
bd   49 288  96 1 "rub" 5 9 10 11
xm   49 20
// reset origin
o 0 0
// draw cursor
m19
// link to next screen
m11 21
#end */

#define keyboard_button_display 17 /*
f 13B
t "`0`" 85 34
#end */

#define keyboard_key 18 /*  index x y letter
bd `0` `1` `2` 1 "`3`" 11 9 10 11
xa `0` p 19 "`3`"
#end */

#define keyboard_press 19 /* letter
mpush "`0`"
t "`M`_    " 0 57
#end */

#define keyboard_erase 20 /*
mpop 1
m19
#end */

#define splash_charts 21 /*
z
f 24BC
t "Data visualization charts" 28 0
// 5 levelbars
o  43 40
m22 0
```

```
o  96 40
m22 1
o 149 40
m22 2
o 202 40
m22 3
o 255 40
m22 4
// 1 long chart
o 10 140
cd 0 0 0 301 49 1 3 1 100 008 2 F00 2 0F0 2 FFF
p 2
l 302 0 302 50
l 0 50 302 50
o 0 0
// link to next screen
m11 24
m23
#end */

#define levelbar_init 22 /*
ld `0` 0 0 20 80 0 0 1 888 100 F00 65 FF0 50 0F0
p 2
l 21 0 21 81
l 0 81 21 81
lv `0` 0
#end */

#define master_flopper 23 /*
// cycle through a prime (to slide on chart) number of relative randoms.
// this thing updates REALLY fast so we need to slow it down a little.
// 1
w 100
lv 0 `R10:50`
lv 1 `R20:60`
lv 2 `R20:60`
lv 3 `R20:60`
lv 4 `R10:50`
cv 0 `R10:20` `R40:60` `R80:100`
// 2
w 100
lv 0 `R10:50`
lv 1 `R40:80`
lv 2 `R40:90`
lv 3 `R20:80`
lv 4 `R10:50`
cv 0 `R10:30` `R30:50` `R60:90`
// 3
w 100
lv 0 `R20:70`
lv 1 `R40:60`
lv 2 `R60:100`
lv 3 `R20:60`
lv 4 `R20:50`
cv 0 `R20:40` `R40:60` `R60:80`
// 4
w 100
lv 0 `R20:50`
lv 1 `R20:60`
lv 2 `R60:80`
lv 3 `R20:60`
lv 4 `R20:50`
cv 0 `R20:30` `R30:60` `R70:80`
```

```
// 5
w 100
lv 0 `R30:50`
lv 1 `R20:100`
lv 2 `R40:100`
lv 3 `R20:100`
lv 4 `R30:50`
cv 0 `R10:30` `R50:80` `R70:100`
// 6
w 100
lv 0 `R40:80`
lv 1 `R20:100`
lv 2 `R40:100`
lv 3 `R20:100`
lv 4 `R40:80`
cv 0 `R10:30` `R60:90` `R60:100`
// 7
w 100
lv 0 `R20:50`
lv 1 `R30:60`
lv 2 `R60:80`
lv 3 `R30:60`
lv 4 `R20:50`
cv 0 `R10:30` `R30:70` `R60:100`
:repeat
#end */

#define splash_drawing 24 /*
z
f 18BC
t "Fast colorful drawing primitives!" 10 210
m11 25
m7
#end */

#define splash_info 25 /*
z
xi 8 0 0
s 0 1
// change this to "m11 9" to exit the demo instead of restarting it
m11 1
m26
#end */

#define logo_bounce 26 /*
// to the right and down
xi 9 0 0
xi 9 1 2
xi 9 2 4
xi 9 3 6
xi 9 4 8
xi 9 5 10
xi 9 6 12
xi 9 7 14
xi 9 8 16
xi 9 9 18
xi 9 10 20
xi 9 11 22
xi 9 12 24
xi 9 13 26
xi 9 14 28
xi 9 15 30
xi 9 16 32
```

```
xi 9 17 34
xi 9 18 36
xi 9 19 38
xi 9 20 40
xi 9 21 42
xi 9 22 44
xi 9 23 46
xi 9 24 48
xi 9 25 50
xi 9 26 52
xi 9 27 54
xi 9 28 56
xi 9 29 58
xi 9 30 60
xi 9 31 62
xi 9 32 64
xi 9 33 66
// to the right and up
xi 9 34 64
xi 9 35 62
xi 9 36 60
xi 9 37 58
xi 9 38 56
xi 9 39 54
xi 9 40 52
xi 9 41 50
xi 9 42 48
xi 9 43 46
xi 9 44 44
xi 9 45 42
xi 9 46 40
xi 9 47 38
xi 9 48 36
xi 9 49 34
xi 9 50 32
xi 9 51 30
xi 9 52 28
xi 9 53 26
xi 9 54 24
xi 9 55 22
xi 9 56 20
xi 9 57 18
xi 9 58 16
xi 9 59 14
xi 9 60 12
xi 9 61 10
xi 9 62 8
xi 9 63 6
xi 9 64 4
xi 9 65 2
xi 9 66 0
// to the left and down
xi 9 65 2
xi 9 64 4
xi 9 63 6
xi 9 62 8
xi 9 61 10
xi 9 60 12
xi 9 59 14
xi 9 58 16
xi 9 57 18
xi 9 56 20
xi 9 55 22
```

```
xi 9 54 24
xi 9 53 26
xi 9 52 28
xi 9 51 30
xi 9 50 32
xi 9 49 34
xi 9 48 36
xi 9 47 38
xi 9 46 40
xi 9 45 42
xi 9 44 44
xi 9 43 46
xi 9 42 48
xi 9 41 50
xi 9 40 52
xi 9 39 54
xi 9 38 56
xi 9 37 58
xi 9 36 60
xi 9 35 62
xi 9 34 64
xi 9 33 66
// to the left and up
xi 9 32 64
xi 9 31 62
xi 9 30 60
xi 9 29 58
xi 9 28 56
xi 9 27 54
xi 9 26 52
xi 9 25 50
xi 9 24 48
xi 9 23 46
xi 9 22 44
xi 9 21 42
xi 9 20 40
xi 9 19 38
xi 9 18 36
xi 9 17 34
xi 9 16 32
xi 9 15 30
xi 9 14 28
xi 9 13 26
xi 9 12 24
xi 9 11 22
xi 9 10 20
xi 9 9 18
xi 9 8 16
xi 9 7 14
xi 9 6 12
xi 9 5 10
xi 9 4 8
xi 9 3 6
xi 9 2 4
xi 9 1 2
:repeat
#end */

// to debug the mpush/mpop buffer
#define display_memory 27 /*
t "`M`"
#end */
```

```
// to debug poweron macros
#define pontest 28
beep 100
w 500
:repeat
#end

// startup calibration option
#define optional_calibration 29
s 0 1
z
f24B
// whole screen touch area
xs 128 0 0 319 239
// if touched, execute macro 30
xm 128 30
t "Touch screen to calibrate." 20 100
w 1000
t "."
w 1000
t "."
w 1000"
t "."
w 1000
xc 128
m8
#end

// calibrate then run demo
#define tc 30
tc
m10
m12
#end

#define splash_fixed_fonts 31 /*
z
f 24BC
t "Fixed Width Fonts Include:" 20 5
sc 0 0
o 25 40
f 4x6
t "4x6 font: quick brown fox 0123"
f 8x10
t "\n"
f 6x8
t "6x8 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x8
t "8x8 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x10
t "8x10 font: quick brown fox 0123"
f 8x12
t "\n"
f 8x13
t "8x13 font: quick brown fox 0123"
f 8x16
t "\n"
f 8x16
t "8x16 font: quick brown fox 0123"
```

```
f 8x16
t "\n"
f 12x24
t "12x24 font: quick 0123"
f 12x24
t "\n"
f 16x32i
t "16x32 font: 0123"
f 8x16
t "\n\n"
f 32x64
t "32X64\xB0C"
m11 16
#end */
```

Appendix D - Tutorial

D.1 *Self-running demonstration*

Once the kit has been assembled per the instructions provided on the CD, the built-in demo routine can be used to verify operation. For the unenclosed kit, install the Jumper JP1 on the PowerCom board and remove any cable from the DB9 connector. For the enclosed kit, attach the supplied loopback plug. Then disconnect / reconnect power to the barrel connector. The display should show "Touch screen to calibrate…" which indicates the demo is operating. Follow the prompts to run the demo.

Note that the Jumper JP1 / loopback plug loops back the SLCD serial data transmit and receive signals to the SLCD. When the SLCD first powers on, it issues a ">" prompt. If it sees that prompt come back, it starts the demo routine.

D.2 *Connection and control via PC*

This section describes how to connect to and control the SLCD from a PC type computer. Any other computer (Mac / Unix / Linux) can be used instead with analogous procedures.

The DB9 connector on the PowerCom board is wired to be compatible with the PC 9 pin serial standard. You need a DB9 female to DB9 male 1-1 serial cable to connect to the PC serial port. Alternatively if you have a USB-serial adapter you can plug the adapter directly onto the PowerCom board.

This tutorial assumes only a basic PC installation is available and therefore uses Hyperterminal to communicate. The program Procomm Plus from Symantec is recommended for advanced work as it supports scripting and other options. See http://www.symantec.com/procomm/

Once the SLCD has been connected to an available serial port, open Hyperterminal (Programs->Accessories->Communications->Hyperterminal) and enter SLCD for the name of the connection. Then enter the serial port connected to the SLCD in the "Connect using" field. Finally, set the Bits per second to 115200, and Flow control to Xon / Xoff. Hit OK and the program main screen appears. Hit the enter (return) key and you should see a ‘>’ prompt character. This indicates that you are communicating with the SLCD board.

Now, go to menu File->Properties->Settings. Set Emulation to TTY. Press the "ASCII Setup", and set "Send line ends with line feeds", "Echo typed characters locally" (i.e. half duplex mode), and "Append line feeds to incoming line ends". Hit OK, OK to return to the main screen.

Now, type ‘z’ followed by the enter key. The display should clear as a result. You should also see the ‘z’ letter you typed and the ‘>’ prompt. You are successfully controlling the SLCD now.

When you want to run the Reach supplied BMPload program, you will need to logically disconnect Hyperterminal from the serial line. To do so, click on the icon showing a telephone with the handset and a small red arrow pointing down, or through the menu Call -> Disconnect. Reconnect again when BMPload is terminated.

D.3 *Simple commands*

This section presents some simple commands that illustrate some of the SLCD capabilities. It assumes that the bitmaps and macro files that were loaded from the factory are still present.

Type in the line(s) as shown in `courier typeface` followed by the enter key. [Note: to minimize typing, you can use the "Text select tool" in Acrobat Reader to select each line, right click to copy, then right click in Hyperterminal and choose "Transmit to Host"]

Clear the screen:

```
z
```

Type Hello World in a 24 point bold font starting at pixel x=100, y=110:

```
f 24B
t "Hello World" 100 110
```

Same as above, but with yellow text on a blue background:

```
z
f 24B
s 16 2
t "Hello World" 100 110
```

Create a vertical blue rectangle at x=40, y=100 to x = 60, y = 150.

```
z
s 2 1
r 40 100 60 150 1
```

Restore fore / back color to black on white:

```
s 0 1
z
```

Alternative way to do the blue rectangle without changing the foreground color:

```
z
r 40 100 60 150 1 00F
```

Display stored full screen bitmap:

```
xi 7 0 0
```

Define momentary button #1 named "Test" in the middle of the screen that sends a return string when pressed and released:

```
z
f 16B
bd 1 150 110 5 "Test" 2 8 10 11
```

D.4 *Macros*

The SLCD comes with pre-loaded macros to demonstrate this capability. Refer to Section E.6 above, or the file "Macros.mac" on the distribution CD.

Enter the following command to invoke the top level macro to display a keypad and display the last number pushed in an entry box:

```
m6
```

Macros have a repeat capability allowing them to loop while waiting for a button to be pressed that will jump to another macro. This is how the demo is implemented. To break out of repeating macros, hit the Escape key followed by Enter.

D.5 *Developing your Application*

Developing your application involves creating as many different screen pages as you need. For each page:

1. Design the bitmaps you want to use using a graphics editor. You can use Adobe Photoshop®, Photoshop Lite, GIMP (Open Source), or Windows Paint to create the bitmaps. See Appendix H.
2. Create a 320x240 pixel canvas using the above, and place the bitmaps where you want them to go. The graphics editor can be used to determine the top right point of the bitmap in terms of X, Y pixels. This is used in the SLCD command to locate the image and text.
3. Download the bitmaps using BMPload.
4. Write a series of SLCD commands to build the display screen and process the defined buttons.

Application note AN-100 describes an example program written for the Rabbit / Zworld RCN3720 core module. It is a useful starting point for developing SLCD control programs.

Appendix E – Working with bitmaps

E.1 *Creating bitmaps*

Bitmaps are used to create the visual elements of the user interface. These include buttons, tabbed folders, and data entry and display areas. Most interface styles implemented on Microsoft Windows can be duplicated on the SLCD. The way to do this is to create or capture the visual element and create the desired layout.

The popular programs used to create bitmaps (.BMP files) are Adobe Photoshop, and the open source program, GIMP.

To capture any visual element on the PC screen, hit the "PrintScreen" button on the PC's keyboard. This captures the screen to the clipboard. Then open the image editing program, open a new window, and paste the screen to that window. The desired elements can then be copied and saved.

Note that bitmaps must be saved in indexed color mode. In this mode, an 8 bit (256 entry) palette maps 8 bit color indices to screen colors.

E.2 *Color Palette*

The SLCD has a fixed 8 bit palette of 232 colors. While the bitmap loader can load a bitmap with any palette, and the SLCD can display any bitmap, they are displayed more quickly if the bitmap palette is the same as the SLCD’s. One way to do this is save the bitmap using the SLCD’s palette. The SLCD palette in Photoshop palette file format is provided on the CD as the file ps8666.act. To use it, in Photoshop, select Image from the top level menu, and then follow:

Image->Mode->Indexed Color->Palette Custom

And load the ps866.act file.

This will convert the working bitmap into the native colors of the SLCD.

Appendix F – RS485 Multipoint Communications

F.1 *Overview*

The SLCD board Revision G does not have RS485 as a physical interface option. However an external RS232 to RS485 converter can be used. This type of adapter automatically enables the RS485 transmitter when the RS232 transmit data is active. Either half duplex or full duplex RS485 can be supported.

In order to support multipoint communications, the Version 2.3.0 and above software has an option to support addressed polling. This forces all SLCD responses including button pushes to be queued and reported only with a poll command. This appendix describes how to use this protocol.

The protocol supports a maximum of 255 SLCD controllers on a shared line; the actual limit may be less than this due to physical bus loading limitations.

F.2 *Setup*

A setup command is used to place the unit into RS485 mode. This mode is saved in non-volatile memory and will remain enabled unless explicitly disabled. Once enabled, the SLCD will not respond to commands unless they are preceded by the RS485 address header.

Setup command:

```
*rs485 <SOF><AD1><AD2><return>
```

`<SOF>` single ASCII character to be used as the "Start Of Frame" character for the shared communication bus. This should not be the '>' character, and must be unique so that it is not used for anything except the start of frame.

`<AD1>` single ASCII character from '0' to '9' and 'A' to 'F' which is the most significant address character.

`<AD1>` single ASCII character from '0' to '9' and 'A' to 'F' which is the least significant address character.

NOTE: address FF is reserved for the host address.

Example:

```
*rs485 /12<return>
```

This sets the 485 mode and specifies '/' as the SOF character and address hex 12 (equivalent to decimal 18) as the unit address. Note that if the character '/' will be used in a text command to the SLCD, then another character such as '`' (backtick) should be used as the SOF.

For the example above, the SLCD responds as follows:

```
RS485 Mode SOF 0x2F (/) ADR 12<return>
```

This response verifies the setup since from this point onwards the SLCD will use these selections for addressing.

F.3 *Command Operation*

Once in rs485 mode, all commands to the SLCD must start with the three character address prefix specified in the setup command, and the selected SOF character should not be used within the command itself. Otherwise, the command syntax is the same as non rs485 mode. The unit responds to commands exactly the same as normal mode except that all responses start with the three character prefix <SOF>FF. The FF address is reserved for the address of the host on the rs485 bus.

Examples:

```
SEND:      /12z<return>
RECEIVE:   /FF>
```

F.4 *Button responses and polling*

All messages from the SLCD that are caused by button presses (for example button notification and macro execution messages) are queued in the order they occurred and are sent when the host next initiates communication with the unit. This includes the poll command which is a null command - the three character prefix followed by a <return>. If the host happens to issue a command (for example to change a value on the display) and a button is simultaneously pushed, the host will receive the button notification message before the command completed response.

Polling example: (button 1 pushed)

```
SEND:      /12<return>
RECEIVE:   /FFx1<return>
```

Button response during display command example: (button 1 pushed)

```
SEND:      /12t "12:15pm"<return>
RECEIVE:   /FFx1<return>
RECEIVE:   /FF><return>
```