SLCD+, SLCD6, SLCD43
Software Command Reference Manual

SLCD+, SLCD6, SLCD43 Firmware Version 2.7.0
BMPload Version 1.9.0

4/24/2009

Note: the software included with this product is subject to a license agreement as described in this Manual.

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

Table of Contents

0. HARDWARE LIMITED WARRANTY AND SOFTWARE LICENSE AGREEMENT

0.1. *Hardware Limited Warranty*

REACH TECHNOLOGY, Inc. warrants its hardware products to be free from manufacturing defects in materials and workmanship under normal use for a period of one (1) year from the date of purchase from REACH. This warranty extends to products purchased directly from REACH or an authorized REACH distributor. Purchasers should inquire of the distributor regarding the nature and extent of the distributor's warranty, if any. REACH shall not be liable to honor the terms of this warranty if the product has been used in any application other than that for which it was intended, or if it has been subjected to misuse, accidental damage, modification, or improper installation procedures. Furthermore, this warranty does not cover any product that has had the serial number altered, defaced, or removed. This warranty shall be the sole and exclusive remedy to the original purchaser. In no event shall REACH be liable for incidental or consequential damages of any kind (property or economic damages inclusive) arising from the sale or use of this equipment. REACH is not liable for any claim made by a third party or made by the purchaser for a third party. REACH shall, at its option, repair or replace any product found defective, without charge for parts or labor. Repaired or replaced equipment and parts supplied under this warranty shall be covered only by the unexpired portion of the warranty. Except as expressly set forth in this warranty, REACH makes no other warranties, expressed or implied, nor authorizes any other party to offer any warranty, including any implied warranties of merchantability or fitness for a particular purpose. Any implied warranties that may be imposed by law are limited to the terms of this limited warranty. This warranty statement supercedes all previous warranties, and covers only the Reach hardware. The unit's software is covered by a separate license agreement.

0.2. *Returns and Repair Policy*

No merchandise may be returned for credit, exchange, or service without prior authorization from REACH. To obtain warranty service, contact the factory and request an RMA (Return Merchandise Authorization) number. Enclose a note specifying the nature of the problem, name and phone number of contact person, RMA number, and return address.

Authorized returns must be shipped freight prepaid to Reach Technology Inc. 842 Boggs Avenue, Fremont, California 94539 with the RMA number clearly marked on the outside of all cartons. Shipments arriving freight collect or without an RMA number shall be subject to refusal. REACH reserves the right in its sole and absolute discretion to charge a 15% restocking fee, plus shipping costs, on any products returned with an RMA.

Return freight charges following repair of items under warranty shall be paid by REACH, shipping by standard ground carrier. In the event repairs are found to be non-warranty, return freight costs shall be paid by the purchaser.

0.3. *Software License Agreement*

PLEASE READ THIS SOFTWARE LICENSE AGREEMENT CAREFULLY BEFORE DOWNLOADING OR USING THE SOFTWARE .

This License Agreement (“Agreement”) is a legal contract between you (either an individual or a single business entity) and Reach Technology Inc. (“Reach”) for software referenced in this guide, which includes computer software and, as applicable, associated media, printed materials, and “online” or electronic documentation (the “Software”).

BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT INSTALL OR USE THE SOFTWARE. IF YOU HAVE PAID A FEE FOR THIS LICENSE AND DO NOT ACCEPT THE TERMS OF THIS AGREEMENT, REACH WILL REFUND THE FEE TO YOU PROVIDED YOU (1) DO NOT INSTALL THE SOFTWARE AND (2) RETURN ALL SOFTWARE, MEDIA AND OTHER DOCUMENTATION AND MATERIALS PROVIDED WITH THE SOFTWARE TO REACH TECHNOLOGY INC AT: REACH TECHNOLOGY INC., 842 BOGGS AVE, FREMONT, CALIFORNIA 94539.

Reach Technology Inc. ("Reach") and its suppliers grant to Customer ("Customer") a nonexclusive and nontransferable license to use the Reach software ("Software") in object code form on one or more central processing units owned or leased by Customer or otherwise embedded in equipment provided by Reach.

EXCEPT AS EXPRESSLY AUTHORIZED ABOVE, CUSTOMER SHALL NOT: COPY, IN WHOLE OR IN PART, SOFTWARE OR DOCUMENTATION; MODIFY THE SOFTWARE; REVERSE COMPILE OR REVERSE ASSEMBLE ALL OR ANY PORTION OF THE SOFTWARE; OR RENT, LEASE, DISTRIBUTE, SELL, OR CREATE DERIVATIVE WORKS OF THE SOFTWARE.

Customer agrees that aspects of the licensed materials, including the specific design and structure of individual programs, constitute trade secrets and/or copyrighted material of Reach. Customer agrees not to disclose, provide, or otherwise make available such trade secrets or copyrighted material in any form to any third party without the prior written consent of Reach. Customer agrees to implement reasonable security measures to protect such trade secrets and copyrighted material. Title to Software and documentation shall remain solely with Reach.

SOFTWARE LIMITED WARRANTY. Reach warrants that for a period of ninety (90) days from the date of shipment from Reach: (i) the media on which the Software is furnished will be free of defects in materials and workmanship under normal use; and (ii) the Software substantially conforms to its published specifications. Except for the foregoing, the Software is provided AS IS. This limited warranty extends only to Customer as the original licensee. Customer's exclusive remedy and the entire liability of Reach and its suppliers under this limited warranty will be, at Reach's option, repair, replacement, or refund of the Software. In no event does Reach warrant that the Software is error free or that Customer will be able to operate the Software without problems or interruptions.

This warranty does not apply if the software (a) has been altered, except by Reach, (b) has not been installed, operated, repaired, or maintained in accordance with instructions supplied by Reach, (c) has been subjected to abnormal physical or electrical stress, misuse, negligence, or accident, or (d) is used in ultrahazardous activities.

DISCLAIMER. EXCEPT AS SPECIFIED IN THIS WARRANTY, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS, AND WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE, ARE HEREBY EXCLUDED TO THE EXTENT ALLOWED BY APPLICABLE LAW.

IN NO EVENT WILL REACH OR ITS SUPPLIERS BE LIABLE FOR ANY LOST REVENUE, PROFIT, OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL, OR PUNITIVE DAMAGES HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE EVEN IF REACH OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

In no event shall Reach's or its suppliers' liability to Customer, whether in contract, tort (including negligence), or otherwise, exceed the price paid by Customer. The foregoing limitations shall apply even if the above-stated warranty fails of its essential purpose. SOME STATES DO NOT ALLOW LIMITATION OR EXCLUSION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES.

The above warranty DOES NOT apply to any beta software, any software made available for testing or demonstration purposes, any temporary software modules or any software for which Reach does not receive a license fee. All such software products are provided AS IS without any warranty whatsoever.

This License is effective until terminated. Customer may terminate this License at any time by destroying all copies of Software including any documentation. This License will terminate immediately without notice from Reach if Customer fails to comply with any provision of this License. Upon termination, Customer must destroy all copies of Software.

Software, including technical data, is subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Customer agrees to comply strictly with all such regulations and acknowledges that it has the responsibility to obtain licenses to export, re-export, or import Software.

This License shall be governed by and construed in accordance with the laws of the State of California, United States of America, as if performed wholly within the state and without giving effect to the principles of conflict of law. If any portion hereof is found to be void or unenforceable, the remaining provisions of this License shall remain in full force and effect. This License constitutes the entire License between the parties with respect to the use of the Software.

1. SLCD/6/43 Operational Overview

1.1. *General*

This manual pertains to the SLCD+, SLCD6, and SLCD43 controllers. These will be referred to generically as the SLCD controller. The main differences between these controllers are:

SLCD43:

- hi color only, 480x272 pixels

SLCD+, SLCD6:

- 8 bit or hi color, 320x240 pixels

The SLCD plus attached LCD touch panel acts as a "smart terminal" and is generally connected to a "host" processor that implements the desired Graphical User Interface (GUI). The host can be any kind of processor from an eight bit microcontroller to a PC. The host issues commands to the SLCD and receives button press responses from the SLCD. In this manual, the term "host" is used to describe the device connected to the SLCD.

The SLCD board contains flash memory that is used for bitmap and macro storage. (This is sometimes referred to as “external” flash to distinguish it from the processor’s internal flash memory that stores the SLCD processor firmware.) A bitmap is equivalent to a Windows ™ bitmap file – it is a rectangular image. Section 3 describes the BMPload program used to store these into the SLCD. Macros are a sequence of SLCD commands and are described in Section 4.

The SLCD is connected to the host processor via a serial port. The number of serial ports is SLCD model-specific; refer to the board reference manual for each model. There are several reasons for having multiple ports:

a) Host program development and debugging. One port is connected to the host and another to a PC. The PC is used to download images and macros that the host uses. The two ports allow both the host and PC to be connected without having to switch cables. The PC can also be used for interactive command execution / testing.

b) The SLCD supports serial pass-through via the "aout" and "ain" commands. This allows serial peripherals to be attached to the SLCD and accessed by the host.

1.2. *Bitmaps*

The look and feel of the interface is created by designing bitmaps (.bmp files) that are used for backgrounds, buttons, and controls. These are designed using a graphics design program such as

Adobe PhotoShop or the Open Source GIMP program. As noted above, the SLCD+ and SLCD6 support 8-bit or high color bitmaps, whereas the SLCD43 only supports high color bitmaps. For most new designs, the high color bitmaps should be used. *Note that high color bitmaps are designed in 24 bit color space, and the BMPload program converts them into the more space-economical 16 bit (RGB565) format.* For more on bitmaps, see Section 8.

1.3. *Communications Interface*

General

- Default communication is at a baud rate of 115200 with no parity, software (XON/XOFF) flow control[1], 8 bits of data, and 1 stop bit. The baud rate can be set to a different initial value on power-on by using the POWER-ON MACRO feature.
- Characters are not echoed and all responses end only with a <return> character (0x0d). This is done to maximize communications line efficiency. To use with terminal emulators such as Hyperterminal, select "half duplex" to echo characters locally and "append LF to CR" to add a line feed to the received Carriage Return.
- 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>`.
- Screen pixel x and y values start at the upper left-hand corner. This is, point x=0, y=0. The lower right corner is point x =(pixel width - 1), y=(pixel height - 1).
- The maximum length of any ASCII command including the termination character is 127 characters.
- The SLCD has a USB slave serial port, implemented via the FTDI FT232R chip. The VCOM drivers make this port look like a standard COM port to the PC. Drivers are available at http://www.ftdichip.com/Drivers/VCP.htm. Note the serial port must be COM9 or less to work properly (Windows issue); the com port can be changed under the advanced driver properties in Control Panel->System->Device Manager->USB Serial Port->Properties->Port Settings->Advanced.

[1] Note: Flow control is supported on SLCD receive only; that is, the SLCD transmits XON and XOFF to control the receive data pacing, but does not respond to XON or XOFF control bytes from the host.

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. Here is an example:

  `p2<return>`

  The above command sets the line width to 2.

1.4. *SLCD Input Buffer Processing*

Input Buffer

The SLCD has a 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 SLCD 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 SLCD will issue a prompt for each command after it executes. These prompts may be issued while the host is sending a command to the SLCD (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. If interface drawing speed is not an issue, this is the recommended method.

The SLCD controller issues a prompt when it has finished processing a command. This includes the null command which is a <return>. The null command can always be used to detect the presence and state of the SLCD.

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:

Loop:
- send a null command (single <return> character)
- wait at least 10ms (or longer if the baud rate is very slow)
- look for a received success prompt
- if 3 null commands have been sent without a response, send the abort cmd (*abt<return>) and look for a received success prompt

If the SLCD comes up in the middle of receiving the null command, it may issue a failure prompt, but will issue a success prompt the next time it gets the null command.

Communications Reliability

Communications reliability is very important in an embedded system. If, for any reason, there is failure to communicate, the host can send the SLCD a null command and expect either an error or success prompt. If an error is received, a second null command will generate the success prompt. An error indicates either a high or low level protocol problem.

NOTE: as of version 2.6.29, the SLCD can accept a command with a CRC prefix to verify the command is received exactly as sent by the host (see USING CRC'D COMMANDS for details).

Flow Control

The SLCD implements receive 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 never 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 will not 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 SLCD. 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 SLCD. The limit event can occur when the buffer is full of unexecuted commands.

In both of the previous cases, when the SLCD 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 SLCD can issue the following prompts. Each prompt starts a new response; that is, it follows a previously sent <return>.

- '>'<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

- '#'<return> Indicates that a command CRC error occurred (see USING CRC'D COMMANDS for details)

- ':'<human-readable text><return>
  Provides human-readable error information. Any prompt that starts with a colon can be discarded by the host.

1.5. *Touch interface*

The SLCD contains a touch controller that interfaces to a four wire resistive touch screen. 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

The term “hotspot” refers to an area of the display that is touching 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.

1.6. *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. What is guaranteed is that the messages arrive complete and do not overwrite each other. The debounce timer for touch processing ensures that the host is not overwhelmed by touch notifications.

1.7. *Control port autoswitch*

For added flexibility, the main control port of the SLCD can be switched on the fly to any of the other serial ports. This is called “control port autoswitch”. It is effected by sending four auxEscape characters on the port to be switched to. These characters must not be interrupted by any other auxiliary IO. The first three characters cause the control port to switch and the last generates a success prompt on the new control port. The auxEscape character is the <return> character by default and is programmable via the SET AUX ESCAPE command.

2. SOFTWARE COMMAND REFERENCE

COMMANDS BY FUNCTION:

Function	Command Name	Command
Alignment, Calibration, & Defines	Set Touch Switch Debounce♦	`*debounce <delay in ms>`
	Set Typematic Parameters♦	`typematic <delay> <repeat>`
	Touch Calibrate♦	`tc`
	Reset Touch Calibration	`*RT`
	Set Orientation♦	`*orient [0|1]`
Screen, LEDs Lighting, & Brightness	Display On/Off	`v <on|off>`
	Backlight Brightness Control♦	`xbb [+|-]<level>`
	Backlight Brightness Control	`xbbs [+|-] <level>`
	External Backlight On/Off	`xbl <on|off>`
	Set LED	`led [0|1]`
Text	Set Cursor	`sc x y`
	Define Displayable Cursor	`curs <bitmap index> <transparent color index>`
	Set Displayable Cursor Position	`curp x y`
	Touch Displayable Cursor	`curt`
	Set Text Alignment	`ta [L|C|R][T|C|B]`
	Set Text Mode	`tm [R|T|X|TR|N]`
	Text Display	`t "text string" x y [R|T|X|TR|N]` `t "text string"`
	Text Flashing Display	`tf <index> <t> "text string" x y [R|T|X|TR]`
	Text Flashing Disable	`tfd <index> <state>`
	Text Flashing Enable	`tfe <index>`
	Text Flashing Delete	`tfx <index>`
	Text Flashing Synchronize	`tfs`
	Set Font	`f <fontName>`
	UTF8 Enable / Disable	`utf8 [on|off]`

♦ *Indicates that the command stores the setting in EEPROM (non-volatile)*

Shapes / Pixels	Draw Circle	`c x0 y0 r [f]`
	Draw Line	`l x0 y0 x1 y1`
	Draw Point	`dp x y`
	Draw Rectangle	`r x0 y0 x1 y1 [style] [color]`
	Draw Triangle	`tr x0 y0 x1 y1 x2 y2 [RGB]`
	Draw Outline Polygon	`pg x0 y0 [x/y x/y...]`
	Draw Filled Polygon	`pf x0 y0 [x/y x/y ...]`
	Draw Rotated Polygon	`pgr <angle> x0 y0 [x/y x/y...]`
	Draw Rotated, Filled Polygon	`pfr <angle> x0 y0 [x/y x/y...]`
	Redraw Rotated Polygon	`ppgr <angle> x0 y0`
	Draw Polyline	`pl x0 y0 [x/y x/y...]`
	Draw Rotated Polyline	`plr x0 y0 [x/y x/y...]`
	Draw Filled Ellipse	`ef x y <x radius> <y radius>`
	Draw Ellipse	`e x y <x radius> <y radius>`
	Draw Arc Segment	`a x0 y0 <radius> <start> <end>`
	Pixel Read	`pr`
	Pixel Write (8 bit color)	`pw x y [palIdx]`
	Pixel Write (high color)	`pw x y [RGB 565]`
Draw Settings	Set Color	`s <fore> <back>`
	Set Draw Mode	`d [n|x]`
	Set Origin	`o <x> <y>`
	Set Pen Width	`p <pixels>`
Clear / Unclear	Clear All Touch	`xc all`
	Clear Screen	`z`
	Clear Screen Special	`zs <bitmap index>`
	Clear Touch button / hotspot	`xc <n>`
	Screen Blank (basic)	`sb color`
	Screen Blank (complete)	`SB <color_detail>`
	Screen Unblank (basic)	`su`
	Screen Unblank (complete)	`SU`

Macros	Macro Execute	`m <n> [macro parameters . . ]`
	Macro Abort	`*abt`
	Macro Notify	`*macnote <0|1|2|3>`
	Touch Macro Assign	`xm <touch index><macro index | name> [<macro2 index>]`
	Touch Macro Assign Quiet	`xmq <touch index> <macro index | name> [<macro2 index>]`
	Touch Macro Assign With Parameters	`xa[q] <n> action <macro index | name> <args>`
	Set Power-On Macro♦	`*PONMAC <index | name> [<option>]`
	Set Demo Macro♦	`*DEMOMAC <index | name>`
	Set Variable	`set <internal variable name> <value>`
	Get Variable	`get <internal variable name>`
	Demo	`Demo`
	List Macros Detail	`lsmac`
	Debug Macro	`*macdebug <0|1>`
Animation:	Animation Clear (delete)	`anic`
	Animation Define	`ani <n> <text string>`
	Animation Disable	`anid <n> <yield #>`
	Animation Enable	`anie <n>`
	Animation List	`ani? <n>`
	Animation Synch	`anis`
	Animation Yield	`y <milliseconds> | stop`
	Wait Vertical Retrace	`wvr <line> [<line2>]`
Sound:	Alarm	`al <alarm> <count>`
	Beep Frequency♦	`bf [<hertz>]`
	Beep Once	`beep <count>`
	Beep Repeat	`rb <on> <off> [alarm]`
	Beep Touch	`bb <number>`
	Beep Volume♦	`bv [+|-]<level>`
	Beep Wait	`beepw <count>`
List Commands	List Bitmaps Detail	`lsbmp`
	List Downloaded Records	`ls`
	List Macros Detail	`lsmac`
Levelbars	Levelbar Define	`ld n x0 y0 x1 y1 or inv bv bc <levels>`
	Levelbar Value	`lv n val`

Sliders & Scrolling	Scroll Screen Area	`k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]`
	Slider Define	`sl idx bg x y slider off ornt inv cont hi lo`
	Slider Value	`sv idx val`
Charts	Chart Bitmap Define	`cdb n x y dw bv tv bitmap <pens>`
	Chart Define	`cd n x0 y0 x1 y1 t dw bv tv bc <pens>`
	Chart Values	`cv n pen0_value [pen1_value ..]`
Meters	Meter Define	`md <id> <bitmap> <x> <y> <type> <minVal> <maxVal> <initial_value> <minAngle> <maxAngle> <x0, y0 … [x10, y10]>`
	Meter Value	`mv <id> <value>`
Buttons / Touch	Button Define – Latching State	`bd <n> x y type "text0" "text1" dx0 dy0 dx1 dy1 bmp0 bmp1`
	Button Define – Momentary	`bd <n> x y type "text" dx dy bmp0 bmp1`
	Button Define Center Text	`bdc <n> x y type "text0" ["text1"] [bmp0 bmp1]`
	Define Hotspot	`x <n> x0 y0 x1 y1`
	Define Special Hotspot (Invisible)	`xs <n> x0 y0 x1 y1`
	Define Typematic Touch Area	`xt <n> x0 y0 x1 y1`
	Button Clear	`bc <n>`
	Clear Touch	`xc <n>`
	Clear All Touch	`xc all`
	Disable Touch	`xd <n>`
	Enable Touch	`xe <n>`
	Set (Latching) State Button	`ssb <n> state`
	Set Touch Characteristics	`xset [+|-][p|r|t|T|1..5]`
	Touch Calibrate♦	`tc`
	Set Typematic Parameters	`typematic <delay> <repeat>`
Reset	Reset Board to Manufactured State♦	`*MFGRESET`
	Reset Board / Firmware	`*RESET`
	Reset Touch Calibration♦	`*RT`
Images / Splash Screen	Display Bitmap Image	`xi <index> x y`
	Display Clipped Bitmap Image	`xic <index> x y x0 y0 x1 y1`
	Display Windowed Bitmap Image	`xio <bitmap index> <x> <y> <0|1> <length> <offset>`

	Display OEM Bitmap Image	`i <number> x y`
	Splash Screen♦	`*SPL <number>`
Save / Restore	Restore Drawing Environment	`sr`
	Save Drawing Environment	`ss`
	Window Save	`ws x0 y0 x1 y1 [index]`
	Window Restore	`wr x y [index]`
	Window Restore Rectangle	`wrr x y <width> <height> <index> [<address>]`

♦ *Indicates that the command stores the setting in EEPROM (non-volatile)*

Timing	Wait	`w <number of milliseconds>`
Baud Rates	Set Baud Rate ComN (N = 0, 1, 2, or 3 for SLCD6/43) (N = 0, or 1 for SLCD)	`baudN [230400 | 115200 | 57600 | 38400 | 19200 | 9600]`
Com port I/O	Output String (Main)	`out "<text string>"`
	Output String (Aux)	`aout "<text string>"`
	Write to Aux Port N	`aoutN "<text string>"`
	Read from Aux port N (N = 0, 1, 2, or 3 for SLCD6/43) (N = 0, or 1 for SLCD)	`ainN`
Notifications	Binary Notification Mode	`*binr <0|1>`
Debug	Debug Command	`*cmddebug <0|1>`
	Debug Macro	`*macdebug <0|1>`
	Debug Touch	`*debug <0|1>`
	Color Test	`*TESTC`
Read / Return Commands	CRC External Flash	`*CEXT`
	CRC Processor Code	`*CSUM`
	CRC Screen	`*CRC`
	EEPROM Read	`*eer <hex location>`
	Get Panel Name	`*panel`
	Read Frame Buffer Line	`*FB <line>`
	Read LCD Controller	`XR <hex register>`
Write Commands	EEPROM Write	`*eew <hex location> <hex value>`
	Write LCD Controller	`XW <hex register> <hex value>`

Download	Binary Download	bdld <index> <address> <size> <timeout>
Port Configuration	Set Control Port♦	*com[0|1|2|3]main
	Control Port Autoswitch♦	(see description)
	Set Aux Escape♦	*auxEsc <hex value of ASCII character>
	Set Previous Control Port	*prevCons
SLCD Information	Get Panel Type	*panel
	Version	vers

♦ *Indicates that the command stores the setting in EEPROM (non-volatile)*

COMMANDS IN ALPHABETICAL ORDER

Command Name	Command
Alarm	al <alarm> <count>
Animation Clear	anic
Animation Define	ani <n> <text string>
Animation Delete	anix <n>
Animation Disable	anid <n> <yield #>
Animation Enable	anie <n>
Animation List	ani? <n>
Animation Synch	anis
Animation Yield	y <milliseconds> | stop
Beep Frequency♦	bf [<hertz>]
Beep Once	beep <count>
Beep Repeat	rb <on> <off> [alarm]
Beep Touch	bb <number>
Beep Volume♦	bv [+|-]<level>
Beep Wait	beepw <count>
Binary Download	bdld <index> <address> <size> <timout>
Binary Notification Mode	*binr <0|1>
Button Clear	bc <n>
Button Define Center Text	bdc <n> x y type "text0" ["text1"] [bmp0 bmp1]
Button Define – Latching State	bd <n> x y type "text0" "text1" dx0 dy0 dx1 dy1 bmp0 bmp1
Button Define – Momentary	bd <n> x y type "text" dx dy bmp0 bmp1
Chart Bitmap Define	cdb n x y dw bv tv bitmap <pens>
Chart Define	cd n x0 y0 x1 y1 t dw bv tv bc <pens>
Chart Values	cv n pen0_value [pen1_value ..]
Clear All Touch	xc all
Clear Touch	xc <n>
Clear Screen	z
Clear Screen Special	zs
Clear Touch	xc <n>
Color Test	*TESTC
Control Port Autoswitch	3 consecutive <return> characters to the Aux port
CRC External Flash	*CEXT
CRC Processor Code	*CSUM
CRC Screen	*CRC
Debug Command	*cmddebug <0|1>
Debug Macro	*macdebug <0|1>
	*debug <0|1>

Debug Touch	*debug <0|1>
Define Hotspot	x <n> x0 y0 x1 y1
Define Displayable Cursor	curs <bitmap index> <transparent color index>
Define Special Hotspots (Invisible)	xs <n> x0 y0 x1 y1
Define Typematic Touch Area	xt <n> x0 y0 x1 y1
Demo	Demo
Disable Touch	xd <n>
Display Bitmap Image	xi <index> x y
Display Clipped Bitmap Image	xic <index> x y x0 y0 x1 y1
Display OEM Bitmap Image	i <number> x y
Display Windowed Bitmap Image	xio <bitmap index> <x> <y> <0|1> <length> <offset>
Display On/Off	v <on|off>
Draw Arc Segment	a x0 y0 <radius> <start> <end>
Draw Circle	c x0 y0 r [f]
Draw Ellipse	e x y <x radius> <y radius>
Draw Filled Ellipse	ef x y <x radius> <y radius>
Draw Filled Polygon	pf x0 y0 [x/y x/y ...]
Draw Line	l x0 y0 x1 y1
Draw Outline Polygon	pg x0 y0 [x/y x/y...]
Draw Point	dp x y
Draw Polyline	pl x0 y0 [x/y x/y...]
Draw Rectangle	r x0 y0 x1 y1 [style] [color]
Draw Rotated Polygon	pgr <angle> x0 y0 [x/y x/y...]
Draw Rotated, Filled Polygon	pfr <angle> x0 y0 [x/y x/y...]
Draw Rotated Polyline	plr x0 y0 [x/y x/y...]
Draw Triangle	tr x0 y0 x1 y1 x2 y2 [RGB]
EEPROM Read/Write	*eer <hex location> *eew <hex location> <hex value>
Enable Touch	xe <n>
External Backlight Brightness♦	xbb[s] [+|-]<level>
External Backlight On/Off	xbl <on|off>
Get Panel Type	*panel
Get Variable	get <internal variable name>
Levelbar Define	ld n x0 y0 x1 y1 or inv bv bc <levels>
Levelbar Value	lv n val
List Bitmaps Detail	lsbmp
List Downloaded Records	ls
List Macros Detail	lsmac
Macro Abort	*abt
Macro Execute	m <n> [macro parameters . .]
Macro Notify	*macnote <0|1|2|3>

Meter Define	md <id> <bitmap> <x> <y> <type> <minVal> <maxVal> <initial_value> <minAngle> <maxAngle> <x0, y0 … [x10, y10]>
Meter Value	mv <id> <value>
Output String (Aux)	aout "<text string>"
Output String (Main)	out "<text string>"
Pixel Read	pr
Pixel Write	pw x y [color]
Power-On Macro♦	*PONMAC <index> [<option>]
Read Frame Buffer Line	*FB <line>
Read from Aux port N	ain?
Read LCD Controller	XR <hex register>
Redraw Rotated Polygon	ppgr <angle> x0 y0
Reset Board to Manufactured State♦	*MFGRESET
Reset Board / Firmware	*RESET
Reset Touch Calibration♦	*RT
Restore Drawing Environment	sr
Save Drawing Environment	ss
Screen Blank (basic)	sb color
Screen Blank (complete)	SB <color_detail>
Screen Unblank (basic)	su
Screen Unblank (complete)	SU
Scroll Screen Area	k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]
Set Aux Escape♦	*auxEsc <hex value of ASCII character>
Set Baud Rate port N	baudN [230400|115200|57600|38400|19200|9600]
Set Color (Detailed)	S <fore_detail> <back_detail>
Set Color (basic)	s <fore> <back>
Set Control Port N ♦	*comNmain
Set Cursor	sc x y
Set Demo Macro♦	*DEMOMAC <index>
Set Displayable Cursor Position	curp x
Set Draw Mode	d [n|x]
Set Font	f <type>
Set (Latching) State Button	ssb <n> state
Set LED	led [0|1]
Set Orientation♦	*orient [0|1]
Set Origin	o <x> <y>
Set Pen Width	P <pixels>
Set Previous Control Port	*prevCons
Set Text Alignment	ta [L|C|R][T|C|B]
Set Text Mode	tm [R|T|X|TR|N]
Set Touch Characteristics	xset [+|-][p|r|t|T|1..5]
Set Touch Characteristics♦	*debounce <delay>

Set Typematic Parameters♦	`typematic <delay> <repeat>`
Set Variable	`set <internal variable name> <value>`
Slider Define	`sl idx bg x y slider off ornt inv cont hi lo`
Slider Value	`sv idx val`
Splash Screen♦	`*SPL <index>`
Text Display	`t "text string" x y [R|T|X|TR|N]` `t "text string"`
Text Flashing Display	`tf <index> <t> "text string" x y [R|T|X|TR]`
Text Flashing Disable	`tfd <index> <state>`
Touch Calibrate♦	`tc`
Touch Displayable Cursor	`curt`
Touch Macro Assign	`xm <touch index><macro index | name> [<macro2 index | name>]`
Touch Macro Assign Quiet	`xmq <touch index><macro index> [<macro2 index>]`
Touch Macro Assign w/ Parameters	`xa[q] <n> action <args>`
UTF8 Enable / Disable	`utf8 [on|off]`
Version	`vers`
Wait	`w <number of milliseconds>`
Window Restore	`wr x y [index]`
Window Restore Rectangle	`wrr x y <width> <height> <index> [<address>]`
Window Save	`ws x0 y0 x1 y1 [index]`
Write LCD Controller	`XW <hex register> <hex value>`
Write to Aux Port N	`aoutN "<text string>"`

♦ *indicates that the command stores the setting in EEPROM (non-volatile)*

SET PEN WIDTH

Description	Sets the pen width for line drawing commands including line, rectangle *but not circle*. Reset setting is width of 2.
Command:	`p <pixels>`
Arguments:	`<pixels>` Width of pen in pixels 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. *With no argument, it returns the current drawing mode.*
Command:	`d [n|x]`
Arguments:	n: Normal drawing mode; draws with the colors from SET COLOR command. 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 top, left 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), then the origin can be set before calling the macro, and the compound object will be drawn at that location. 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 (239 portrait) `<y>` Y axis value between 0 and 239 (319 portrait)

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 subsequent commands using a basic color palette. The following assumes the standard palette is used for 8 bit color firmware.

Command — `s` <foreground> <background>

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

<background> = background color value per the table below

Color value	Color
0	Black
1	White
2	Blue
3	Green
4	Cyan
5	Red
6	Magenta
7	Brown
8	Dark Grey

Color value	Color
9	Grey
10	Light Grey
11	Light Blue
12	Light Green
13	Light Cyan
14	Light Red
15	Light Magenta
16	Yellow

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

Example: — `s 0 1`

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

SET COLOR (8 bit color, custom palette)

Description — Sets the background and foreground color for all subsequent commands. Color is defined by palette index values, since with a custom palette there are no fixed colors. The custom palette option is selected via the BMPload program.

Command — `s` <foreground> <background>

Example: — `s 2 3`

This sets the foreground color to palette index value 2, and the background to palette index value 3.

SET COLOR (detailed)

Description	Sets the background and foreground color for all commands using arbitrary RBG values. *Note:*
Command	`S` <foreground_detail> <background_detail>
Arguments:	<foreground_detail> = foreground color value in RGB format
	<background_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
Notes:	1. To reset the background after changing the color, the screen must be cleared using the command, '`z`'.
	2. Using 8 bit color firmware, 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.
	Using high color firmware (SLCD6 /43 only), all 4096 colors are usable.
	3. This command does not function as described when using a custom palette in 8 bit color mode.

SET FONT

Description — Sets the font to be used in subsequent TEXT DISPLAY commands. See BMPload manual entry for downloadable fonts; they simply extend the name space of this command.

Commands:

`f <name>` - set font

`f?` - display list of font names

`f` - display currently active font name

Arguments:

Proportional fonts:
<name> = 8, 10, 10S, 13, 13B, 16, 16B, 18BC, 24, 24B, 24BC, 32, 32B

Fixed width fonts:
<name> = 4x6, 6x8, 6x9, 8x8, 8x9, 8x10, 8x12, 8x13, 8x15B, 8x16, 8x16L, 12x24,14x24, 16x32, 16x32i,

Fixed width, symbol and CAPITALS only fonts:
<name> = 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.

Example:

```
f 13B
```

Set the current font to 13 point bold.

SET UTF8 ENCODING

Description: Enables or disables UTF8 encoding in text strings. When UTF8 is disabled, the 256 character extended ASCII character set per ISO 8859-1 is accessible. This is all that is needed for the basic font set. For downloaded fonts with Unicode sets larger than 256 characters, UTF8 encoding is used.

Command: `utf8 [on|off]`

Example

```
utf8 on
t "\xe4\xb8\x81"
```

This displays the Unicode character 0x4e01 whose UTF8 equivalent is hex E4 B8 81. Note that an embedded host can send the UTF8 characters as three bytes - the text escape is not necessary unless the host can only send 7 bit ASCII.

DISPLAY BITMAP IMAGE

Description: Copies 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/6/43 flash memory. These are accessed by index number.

Command: `xi <index> x y`

Arguments: `<index>` - bitmap index.

`x y` - location of top left corner of bitmap.

Example `xi 4 10 20`

This displays the 4th bitmap at location (10,20).

DISPLAY CLIPPED BITMAP IMAGE

Description: Same as xi command above, except that a clipping area is applied so only part of the bitmap is displayed. This is useful to restore a part of a large graphic that has had text or graphics overlaid on it, for example when a graphic cursor is drawn on a map. When the cursor moves, the map area previously obscured by the cursor needs to be restored. The clip area is defined *relative to the top left of the bitmap* (ie: x0 y0 x1 y1 are offsets from x y). Only uncompressed bitmaps are supported by this command.

Command: `xic <index> x y x0 y0 x1 y1`

Arguments: `<index>` - stored bitmap index.

`x y` - location of top left corner of bitmap

`x0 y0 x1 y1`

- rectangle within the bitmap to be displayed

Note: 0 <= `x0 <` bitmap width, 0 <= `y0 <` bitmap height,
x0 <= `x1 <` bitmap width, y0 <= `y1 <` bitmap height

Example

```
xi 1 10 20     // draw main bitmap
p 1            // set pen width to 1
r 30 40 35 45  // draw rectangle
// calc rectangle offsets (x0 y0 x1 y1):
// (30-10) (40-20) (35-10) (45-10)
xic 1 10 20 20 20 25 25
```

This example draws a main bitmap #1. Then it places a rectangle on top of it. Then, instead of redrawing the entire bitmap to erase the rectangle, the xic command is used to only redraw a part of it.

DISPLAY WINDOWED BITMAP IMAGE

Description: Displays a section (window) of a stored bitmap with an offset. This is used to implement a sliding window into a larger bitmap, for example, a section of a compass or ruler. It can also be used to simulate a rotating dial.

Note that the window is clipped and offset only in the specified direction. For example, with a horizontal compass bitmap, the visible rectangle width is specified with the length parameter, but the height is always the full vertical height of the bitmap.

Command: `xio <index> <x> <y> <0|1> <length> <offset>`

Arguments:

- `<index>` - stored bitmap index.
- `<x> <y>` - screen coordinates for drawing location
- `<0|1>` - 0: Vertical window
  - 1: Horizontal window
- `<length>` - number of pixels to display along orientation
- `<offset>` - offset into

Notes:

1. Only supported on SLCD6 board models.
2. High color firmware will only support high color bitmaps or the xio command will return an error.

Example: `xio 4 10 10 1 100 50`

Bitmap #4 can be wider than the LCD screen; assume it is N pixels long and 45 pixels high. This command draws a rectangular screen area (10,10) to (109, 54) with the source being bitmap #4 with a horizontal offset into the bitmap of 50 pixels.

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|N]
```

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

N – Normal: foreground / background colors are used.

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

The mode (N, R, T, X, TR) most recently specified is used until another mode is specified. Initially, Normal mode is used, and will remain in effect until changed. To prevent confusion about which mode is in effect, always explicitly declare mode.

Examples:

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

This displays the text

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

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

```
t "\xa9Copyright" 0 0 R
t "\n 1999-2009"
```

displays the text

```
©Copyright
 1999-2009
```

at the top left corner of the screen, in Reverse mode

TEXT FLASHING DISPLAY

Description: Displays a flashing text string starting at the current or specified point using the currently set font. The string is alternately drawn in the selected foreground color, then erased with the background color at a rate specified by the parameter <t> (delay time). Each alternate view is displayed for <t> mS. The total time to cycle is 2X the selected delay time. See TEXT DISPLAY for text string escapes and other details.

Command: `tf index t "text string" x y [R|T|X|TR]`

All parameters except index and "text string" are optional. If unspecified, <t> will default to 500 mS. A null text string "" is acceptable.

Arguments: <index> is an identifier for this text; accepted identifiers are 0 through 9.

t is the number of milliseconds between flashes.

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

The only required parameters are the <index> and the "text string". The timing defaults to 500 milliseconds, the text is written to the current cursor position, and the mode will be set to transparent.

Notes: 1. Quotes are required around the text string. ***The entire command including <return> must be less than 120 characters.***

2. The clear screen command 'z' clears all flashing text instances.

Example: `tf 0 300 "FLASHING TEXT" 10 0 T`

This puts the text

```
FLASHING TEXT
```

With the top left corner of the 'P' at location x=10, y=0 with a delay of 300 Milliseconds between displayed and non-displayed text.

TEXT FLASHING DISABLE

Description: Disables flashing text instances as specified by the index (see tf command). The stopping point state can be specified.

Command: `tfd <index> <state>`

Arguments: <index> is the identifier for the text; accepted identifiers are 0 through 9.

<state> specifies the state to stop the animation, for text flash, this is 0 or 1.

Note: To delete and re-use a text flash or animation index, use the "tfd <index> <state>" command to stop the animation at the selected state, and then use the "anix <index>" command to delete the animation.

Examples: `tfd 0 0`

This stops the text flash animation at the first state with text in selected foreground color.

`tfd 0 1`

This stops the test flash animation at the second state with text in the selected background color.

TEXT FLASHING ENABLE

Description: Enables text flashing for individual strings as specified by the identifier. If the text flash animation is currently running for that identifier, no action is performed.

Command: `tfe <index>`

Arguments: <index> is the identifier for the text; accepted identifiers are 0 through 9.

Examples: `tfe 0`

This resumes the text animation from a previously stopped state.

TEXT FLASHING DELETE

Description: Deletes the specified text flash animation.

Command: `tfx <index>`

Arguments: <index> is the identifier for the text; accepted identifiers are 0 through 9.

Examples: `tfx 0`

This stops and deletes from memory the specified (0) text animation.

TEXT FLASHING SYNCHRONIZE

Description:	Synchronizes all animations.
Command:	`tfs`
Arguments:	None
Examples:	`tfs`
	Resets all animations and flashing text to initial state. If animations or flashing text are using integral delays, the animation and text flashing will be performed in synchronization.

TEXT FLASH ANIMATION ENABLE

Description:	Re-Enables currently stopped text flash animation
Command:	`tfe <index>`
Arguments:	<index> is the identifier for the text; accepted identifiers are 0 through 9.
Examples:	`tfe 0`
	Stopped test flash animation is re-enabled and executes.

SAVE DRAWING ENVIRONMENT (State Save)

Description:	Saves the current drawing state including color, pen and line style, cursor position and origin (margins), font, text alignment and drawing mode.
	Note: each Macro has its own memory for state save/restore.
Command:	`ss`
Arguments:	None
Examples:	`ss`
	Save the drawing state.

RESTORE DRAWING ENVIRONMENT (State Restore)

Description:	Restores the drawing state. If the save state (ss) command has not been executed since power up or reset, the power up state is used.
	Note: each Macro has its own memory for state save/restore.
Command:	sr
Arguments:	None
Examples:	sr
	Restore the drawing state.

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. *With no argument, it returns the current cursor location.*

Command: `sc x y`

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

Is equivalent to:

```
t "hello" 10 20
```

DEFINE DISPLAYABLE CURSOR

Description: Defines a cursor (bitmap), with the transparent color of the bitmap. The bitmap is displayed initially at coordinates 0, 0. Once this command is used, the displayed cursor can be acted upon with other "DISPLAYABLE CURSOR" commands.

The DISPLAYABLE CURSOR commands have significant restrictions. Refer to the notes below for use.

Command: `curs <bitmap index> <transparent color index>`

Example: `curs 5 2<return>`

Use bitmap index 5 for the displayable cursor. Use index 2 in the color palette for 8-bit indexed bitmap as the transparent color.

Notes:

1. Only available on the SLCD6 Display Controller.
2. Bitmap must be 8 bits per pixel, indexed.
3. Only first 3 colors in color table are used.
4. The width and height must be a multiple of 16 pixels (e.g. 16x16 or 32x32 or 48x48). This is due to a hardware limitation.
5. Maximum cursor size is 64x64.
6. Bitmap index value 0 is used to clear the cursor.

SET DISPLAYABLE CURSOR POSITION

Description: Sets the displayable cursor at an x-y coordinate.

Command: `curp x y`

Example: `curp 10 20<return>`

This sets the top left corner of the displayable cursor to (10, 20).

TOUCH DISPLAYABLE CURSOR

Description: Simulates a screen touch at the current x-y coordinate of the displayable cursor. This command is only effective if the displayable cursor's position is over a button or hotspot. This is used in a fashion similar to a mouse button press on a Personal Computer.

Command: `curt`

Example: `curt<return>`

Response: `x135<return>`

The displayable cursor was over the area defined by touch button number 135 when the TOUCH DISPLAYABLE CURSOR command was executed.

SET TEXT ALIGNMENT

Description: Sets the alignment of the subsequent text relative to the specified (x, y) location in the SET CURSOR or TEXT DISPLAY command. ***NOTE: horizontal alignment reverts to Left after a text display command is issued.*** If no argument is given, the response is the current alignment.

Command: `ta [L|C|R][T|C|B]`

Arguments: First argument a single letter for horizontal alignment:
L = left
C = center
R = right

Second argument a single letter for vertical alignment:
T = top
C = center
B = bottom

Example1:
```
ta RB
t "hello" 100 110
```

This will place the text to the right and above the point 100, 110.

Example2:
```
ta CT
ta
```

Returns: `CT` (the current alignment mode is returned)

SET TEXT MODE

Description: Sets the text draw mode for subsequent TEXT DISPLAY command. With no argument, the command returns the current mode.

Command: `tm [R|T|X|TR|N]`

Arguments: Same as TEXT DISPLAY, with N for "normal".

DRAW POINT

Description: Draws a point with current pen width and foreground color.

Command: `dp x y`

Example: `dp 50 100`

This will draw a point at x=50, y=100.

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 (in landscape mode).

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 100 100 179 119`

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

`r 100 100 179 119 1`

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

`r 50 100 179 119 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.

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.

Note: To fill without an outline border, set the pen width to 1.

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 cyan.

PIXEL WRITE

Description Sets a single pixel to either the foreground color or a specific palette index value. For mapping of palette index to color, see Appendix H. when running high color firmware, the value is a 16 bit hex value in 565 format RGB.

Command: `pw x y [palIdx]`

Command: `pw x y [RGB]`

Arguments: `x y` – location of pixel
`palIdx` – if provided, palette index to write. If not provided, the foreground color is used.

RGB - color as 565 (RRRRRGGGGGGBBBBB binary) in hex, e.g. F800 is pure red.

PIXEL READ

Description: Returns the 2 character hex pixel palette index value for the specified pixel. In high color mode, returns the 4 character hex color in 565 format.

Command: `pr`

Example: (8 bit color)
```
s 2 3
z
pr 0 0
```

Returns: `1E`

Example: (high color)
```
s 2 3
z
pr 0 0
```

Returns: `07E0`

DRAW OUTLINE POLYGON

Description: Draws a polygon at specified origin.

Command: `pg <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: <X Origin> <Y Origin> is X/Y translation for polygon.

<X/Y vertices ...> are vertex end-points (MAX=11).

Example: `pg 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`

Draws polygon at offset 100 100.

DRAW FILLED POLYGON

Description: Draws a filled polygon at specified origin.

Command: `pf <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: <X Origin> <Y Origin> is X/Y translation for polygon.

<X/Y vertices ...> are vertex end-points (MAX=11).

Example: `pf 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`

Draws filled polygon at offset 100 100.

DRAW ROTATED POLYGON

Description: Draws a rotated polygon at specified origin.

Command: `pgr <Angle> <X Orig> <Y Orig> <X/Y vertices...>`

Arguments: <angle> Number of degrees to rotate polygon

<X Origin> < YOrigin> is X/Y translation for polygon.

<X/Y vertices ...> are vertex end-points (MAX=11).

Example: `pgr 45 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`

Draws polygon rotated 45 Deg. at offset 100 100.

DRAW ROTATED FILLED POLYGON

Description: Draws a rotated, filled, polygon at specified origin.

Command: `pfr <Angle> <X Orig> <Y Orig> <X/Y vertices....>`

Arguments: <angle> Number of degrees to rotate polygon

<X Origin> <Y Origin> is X/Y translation for polygon.

<X/Y vertices ...> are vertex end-points (MAX=11).

Example: `pfr 45 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`

Draws filled polygon rotated 45 Deg. at offset 100 100.

REDRAW ROTATED POLYGON

Description: Draws a rotated, filled, polygon at specified origin.

Command: `ppgr <Angle> <X Orig> <Y Orig>`

Arguments: <angle> Number of degrees to rotate polygon

<Y Origin> <Y Origin> is X/Y translation for polygon.

Example: `ppgr 45 100 100`

Draws previously defined polygon rotated 45 Deg. at offset 100 100.
Defined polygon persists until overwritten by a new polygon definition.

DRAW POLYLINE

Description: Draws a rotated polygon at specified origin.

Command: `pl <X Orig> <Y Orig> <X/Y vertices...>`

Arguments: <X Origin> <Y Origin> is X/Y translation for polygon.

<X/Y vertices ...> are vertex end-points (MAX=11).

Example: `pl 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`

Draws polyline at offset 100 100.

DRAW ROTATED POLYLINE

Description:	Draws a rotated polygon at specified origin.
Command:	`plr <Angle> <X Orig> <Y Orig> <X/Y vertices...>`
Arguments:	<angle> Number of degrees to rotate polygon
	<X Origin> <Y Origin> is X/Y translation for polygon.
	<X/Y vertices ...> are vertex end-points (MAX=11).
Example:	`plr 45 100 100 3 0 4 2 6 3 4 4 3 6 4 2 0 3 2 2`
	Draws polyline rotated 45 Deg. at offset 100 100.

DRAW FILLED ELLIPSE

Description:	Draws a filled ellipse.
Command:	`ef x y <x radius> <y radius>`
Arguments:	x y Specifies the center point of the ellipse.
	<x radius> Specifies the X radius of the ellipse
	<y radius> Specifies the Y radius of the ellipse
Notes:	1. Ellipse radii are limited to values of 180.
	2. Ellipses are limited to horizontal and vertical orientation.
Example:	`ef 150 150 30 50`
	Draws a ellipse centered at 150X150, Ellipse is vertically orientated.

DRAW ELLIPSE

Description:	Draws an ellipse.
Command:	`e x y <x radius> <y radius>`
Arguments:	x y Specifies the center point of the ellipse.
	<x radius> Specifies the X radius of the ellipse
	<y radius> Specifies the Y radius of the ellipse
Notes:	1. Ellipse radii are limited to values of 180.
	2. Ellipses are limited to horizontal and vertical orientation.
Example:	`e 150 150 30 50`
	Draws an ellipse centered at 150X150. Ellipse is vertically orientated.

DRAW ARC SEGMENT

Description:	Draws a Arc Segment.
Command:	`a <X0> <Y0> <Radius> <Start Angle> <End Angle>`
Arguments:	<X0> <Y0> Center point of the ARC segment
	<Radius> Radius of arc (Pixels)
	<Start Angle> Starting angle (Degrees)
	<End Angle> Ending angle (Degrees)
Example:	a `100 100 40 20 110`
	Draws a semi-circle centered at 100X100.

SCROLL SCREEN AREA

Description:	Scrolls a screen area up, down, left, or right. The background color is used to fill in the moved pixels. Can also rotate left or right by one pixel.
Command:	`k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]`
Arguments:	`x0 y0 x1 y1` – defines the rectangle area for the scroll.
	`<numlines>` - number of lines to scroll. Must be 1 for 'L' or 'R' action
	`l` = left scroll; `r` = right scroll; `u` = up scroll; `d` = down scroll
	`L` = left rotate 1 pixel (<numlines> must be 1)
	`R` = right rotate 1 pixel (<numlines> must be 1)
Note:	1. <numlines> is limited to the number of pixels in the axis of the scroll.
	2. E.g. If the rectangle is 10x X 20y pixels, the maximum X <numlines> is 10 and the maximum Y <numlines> is 20.

Example:	`f13B`
	`t "line 1\nline 2" 100 120`
	`k 100 120 140 146 13u`

	This displays 2 lines of text and then scrolls up the text area such that the lower line replaces the upper line.

CHART BITMAP DEFINE

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

Command: `cdb n x y dw bv tv bitmap <pens>`

Arguments:

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

`x and y` are the top left corner coordinates. The bottom right corner coordinate is defined by the width (x axis length) and height (y axis length) of the chart area.

`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)

`bitmap` – bitmap index

`<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 4 0 99 333 2 0FF 1 F00`

Defines a chart in the rectangular area defined by bitmap index 99, starting in the upper left (10,20). The lower right is defined by the horizontal and vertical resolution. Each data value will be 4 horizontal pixels wide. The chart ('Y') values are scaled from 0 to 99. The background bitmap is index 99. Two pens are defined: the first is pen width 2, color teal (0FF), the second is pen width 1, color red (F00).

CHART DEFINE

Description: Creates 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, behavior is determined by chart type:

- 0 (STRIP): current data is shifted left and new data is added at the right hand edge of the chart, like a strip chart recorder.
- 1 (OSCILLOSCOPE): the new data is added at the left edge of the chart, overwriting the oldest data, like an oscilloscope.

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 0 (STRIP) or 1 (OSCILLOSCOPE)

`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)

`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. The second cv command extends the teal pen another 4 pixels in the X+ (left to right) direction and to 50 in the Y axis. The red pen moves 4 pixels in the X+ direction and to 60 in the Y axis.

LEVELBAR DEFINE

Description: Creates 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..

SLIDER DEFINE

Description: Creates a slider object using background and slider control bitmaps.

Command: `sl idx bg x y slider off ornt inv cont hi lo`

Arguments: `idx` - slider index. Must be in the range 128 to 136 (maximum 8 sliders). Note that slider indices are shared with hotspot indices; that is if a slider is defined with index 128, hotspot index 128 cannot be used.

`bg` – background bitmap index

`x, y` - top left corner to place the background bitmap

`slider` – slider control (e.g. knob / button) bitmap index

`off` – slider offset from the edge of the background bitmap

`ornt` – orientation: 0 = vertical; 1 = horizontal

`inv` – invert: 0 = top / left is low; 1 = bottom / right is low

`cont` – continuous touch:
0 = slider cannot be moved by sliding the touch point
1 = slider can be moved by sliding the touch point

`hi` – maximum slider value

`lo` – minimum slider value

Host notification when slider value is changed:

`l<idx>:<value>`

Example: `sl 128 44 100 30 45 5 0 1 1 100 0`

This example assumes that the demo bitmaps are loaded with 44 and 45 being the slider background and control respectively. A slider is created in the middle of the screen (left corner = 100, 30) in a vertical orientation with the control bitmap offset 5 pixels from the left edge of the background bitmap. The touch action is continuous and the slider values range from 0 at the bottom to 100 at the top.

Example notification: `l128:50`

SLIDER VALUE

Description: Sets the value of a previously defined slider object.

Command: s`v idx val`

Arguments: `idx` - slider index

`val` - value for the slider.

Example: `sv 128 50`

Sets slider index 128 to value 50.

CIRCULAR SLIDER DEFINE

Description: Creates a circular slider object using background and slider control bitmaps.

Command: `slc hotspot, background, x, y, slider, type, top, bottom, value, scope, maxAngle, minAngle, startDeg, drawRadius, maxRange`

Arguments:

`hotspot` - slider index. Must be in the range 128 to 136 (maximum 8 sliders). Note that slider indices are shared with hotspot indices; that is if a slider is defined with index 128, hotspot index 128 cannot be used.

`background` – background bitmap index

`x, y` - top left corner to place the background bitmap

`slider` – circular slider control (e.g. knob / button) bitmap index

`type` – 1 =

`top` – maximum slider value

`bottom` – minimum slider value

`value` – value between top/bottom to place slider knob

`scope` – number of value points to change per revolution. This parameter is only valid for type 2 circular sliders. Any value given for type 1 circular sliders will be ignored. The scope must be at least 1 and no larger than the difference between top and bottom.

`maxAngle` – maximum angle (0-360) from positive x-axis in degrees to place slider knob for type 1 sliders. The maximum value will be tied to this angle. Any value given for type 2 circular sliders will be ignored

`minAngle` – minimum angle (0-360) from positive x-axis in degrees to place slider knob for type 1 sliders. The minimum value will be tied to this angle. Any value given for type 2 circular sliders will be ignored

`startDeg` – angle (0-360) from positive x-axis in degrees to place slider knob/ to calculate out of bounds. This parameter is only valid for type 2 circular sliders. Any value given for type 1 circular sliders will be ignored

`drawRadius` – radius from center of the background image to draw the rotator/knob image. The draw radius must be small enough so the rotator/knob will never be drawn outside of the background image.

`maxRange` – maximum range in pixels from the center of the rotator/knob slider that a touch will take effect.

CIRCULAR SLIDER DEFINE (continued)

Host notification when slider value chages begins with 'l' (lower case L):

`l<idx>:<value>`

Notes:

1. For type 1 sliders there will be a dead zone between the min/max angle. Type 2 sliders have accessible dead zones when out of range of the min/max values but the value will not go beyond the min/max.

2. All circular sliders are clockwise increasing. For example, if you have a type 1 slider with min at 0 and the max at 360 the rotator/knob will not move because there are 0 degrees for the rotator to move. Simply switch the min/max values to get the full 360 degrees.

Example: `slc 128 46 100 30 47 1 500 0 250 0 240 300 0 25 120`

This example assumes that bitmaps are loaded with 46 and 47 being the slider background and control respectively. A type 1 slider is created in the middle of the screen (left corner = 100, 30) with possible values between 0 and 500 with its initial value at 250. The minimum value will be tied to 240 degrees and the max at 300 degrees from the positive x-axis. This will make the initial value of 250 (half of the max) to be placed midway between min/max (90 degrees). The rotator/knob will be drawn at 25 pixels from the center of the image with a valid touch being at most 120 pixels away.
Example notification: `l128:50`

`slc 128 46 100 30 47 2 4000 1 2000 1000 0 0 90 25 120`
This example assumes that bitmaps are loaded with 46 and 47 being the slider background and control respectively. A type 2 slider is created in the middle of the screen (left corner = 100, 30) with possible values between 1 and 4000 with its initial value at 1000. This will make the initial value of 2000 tied to 90 degrees form the positive x-axis. The value will change by 1000 every rotation until the min/max has been reached. The rotator/knob will be drawn at 25 pixels from the center of the image with a valid touch being at most 120 pixels away.
Example notification: `l128:50`

CIRCULAR SLIDER VALUE

Description: Sets the value of a previously defined circular slider object.

Command: s`vc idx val`

Arguments: `idx` – circular slider index

`val` - value for the slider.

Example: `svc 128 50`

Sets slider index 128 to value 50.

METER DEFINE

Description: Creates a "Meter" object that resembles an analog meter (with an indicator). The meter object uses a background bitmap that visually represents the meter, and a polygon for the indicator needle.

Command:
```
md <idx> <bitmap> <x> <y> <type> <minVal>
<maxVal> <init_val> <minAngle> <maxAngle> <x0 y0>
<x1 y1>  . . . [x10 y10]>
```

Arguments:

`idx` - meter index. The meter index must be in the range 0 to 7 (maximum 8 meters).

`Bitmap` – background bitmap index

`x, y` – top left corner to place the background bitmap

`type` – always 1.

`minVal` – minimum numerical value for indicator

`maxVal` – maximum numerical value for indicator

`init_val` – initial numerical value for indicator

`minAngle` – minimum angle for minimum numerical value for indicator.

`maxAngle` – maximum angle for maximum numerical value for indicator

`x0 y0` – pivot point for indicator relative to 0,0 top left of bitmap

`x1 y1 . . .[x10, y10]`

– polygon points for indicator relative to pivot point. Max 10 points

Notes: The angle values are with respect to the indicator as specified by the polygon points where 0 is as drawn and positive degrees move to the right (clockwise).

See Draw Rotated Filled Polygon for details on the operation of the indicator needle parameters.

Example:
```
md 1 48 0 0 1 475 515 500 270 90 126 120 -4 0 0 -
78 4 0
```

This example defines a meter with index number 1, using bitmap index 48 as the background image. The type is always 1. The minimum value of 475 for the indicator is at angle 270 degrees (90 degrees to left of vertical), and the maximum value of 515 is at angle 90 degrees. The indicator will point to initial value 500. The indicator pivot point is 126 120 and the indicator is drawn as a vertical triangle with polygon points -4 0 0 -78 4 0.

METER VALUE

Description: Sets the value of indicator for a specified meter. The meter must have been previously created by the Meter Define command.

Command: mv `id value`

Arguments `id` - meter index value previously defined.

`value` - value to set indictaor. Must be in the range of values as defined by the Meter Define command

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.

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 bitmap

`type` Button type:

- 1 Standard. Displays bitmap `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.
- 30 Typematic; same as type 3 above, except that subsequent host notifications do not generate a beep.
- 4 Same as standard, except host is notified only when the button is released.
- 5 Same as standard, with both press and release notification.

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

`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)

Notes: 1. When a button is number is redefined, all macro assignments are cleared.

2. Button numbers 118-127 support “long strings” of a length of 50 characters, rather than the default of 20.

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 bitmap `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

  Note: both bitmaps must be the same size.

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

Note: Button numbers 118-127 support “long strings” of a length of 50 characters, rather than the default of 20.

Example1: `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).

Host notification: `s31<return>` or `s30<return>`

Example2: `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.

BUTTON DEFINE CENTER TEXT

Description: Defines a momentary or latching state touch button on the screen. The text for the button(s) is automatically centered vertically and horizontally. The difference between this command and other BUTTON DEFINE commands syntactically, is that the text offsets are not needed.

Command: `bdc <n> x y type "text0" ["text1"] [bmp0 bmp1]`

Arguments:

`<n>`	Button number, must be in the range of 0 to 127.
`x y`	Upper left hand corner of the button bitmap
`type`	Button type: `All types supported in` BUTTON DEFINE – LATCHING STATE and BUTTON DEFINE – MOMENTARY commands.
`"text0"`	Text string to be displayed on the button. Quotes are required. The current foreground color will be used for the text. For multi-line text, use the newline ('\n') character decimal 10 in the string.
"text1"	Optional argument for BUTTON DEFINE – LATCHING STATE types.
`bmp0`	Optional argument which is index of bitmap displayed in the unpressed state.
`bmp1`	Optional argument which is index of bitmap displayed in the pressed state.

See BUTTON DEFINE – LATCHING STATE and BUTTON DEFINE– MOMENTARY commands for host notification.

Note: Both bitmaps must be the same size.

Example: `bd 23 150 100 1 "Test" 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 vertical and horizontal center 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>`

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. *Note:if a macro is assigned to the button it may be acivated by this command.*
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.
Host notification:	x<n><return> Sent when the corresponding hotspot is pushed. Note that once a button or hotspot is defined, the notification can be transmitted at any time including during a command transmission to the unit (full duplex).
Note:	When creating multiple hotspots keep in mind that pre-existing hotspots may affect operation unless cleared using the "Clear Touch button / hotspot" command.

Example: `x 135 100 100 179 139`

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

Example notification when hotspot is pressed:

`x135<return>`

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 135 100 100 179 139`

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

Note: When creating multiple hotspots keep in mind that pre-existing hotspots may affect operation unless cleared using the “Clear Touch button / hotspot” command.

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.

SET TOUCH CHARACTERISTICS

Description:	Allows a button or hotspot characteristics to be modified after being defined. Useful to make a typematic hotspot.
Command:	`xset` <n> `[+|-][p|r|t|T|1|2|3|4|5]`
Arguments:	`+` (optional) add the following option `-` remove the following option p notify on press r notify on release t typematic T typematic special (beep on first press only) `1..5` (buttons only) set button type

CLEAR TOUCH BUTTON / HOTSPOT

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`

CLEAR SCREEN

Description: Clears the screen to the background color and removes all buttons, hotspots, charts, levelbars, and sliders.

Command: `z`

CLEAR SCREEN SPECIAL

Description: Same function as 'z' command but the display is either not cleared or cleared by writing a full screen bitmap.

Command: `zs`

Clears all buttons, charts etc like 'z' but does not change the display

Command: `zs <bitmap index>`

Same as 'z' but instead of clearing the screen, it displays the specified bitmap at location (0, 0). This is useful when a full screen bitmap is used.

SCREEN BLANK (basic; 8 bit firmware only)

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

SCREEN UNBLANK (basic; 8 bit firmware only)

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

Command: `su`

SCREEN BLANK (complete; 8 bit firmware only)

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 RGBformat

RGBformat = RGB where R, G, B are each a single character from 0 to F. See SET COLOR (detailed) for more information.

Example: `SB 003`

Sets the entire screen to a light blue.

SCREEN UNBLANK (complete; 8 bit firmware only)

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

Command: `SU`

WINDOW SAVE (not available for SLCD or SLCD43)

Description: Saves contents of a rectangular screen area to an off-screen buffer. This command is not available for the WQVGA panels. Maximum size available depends on color depth. See WINDOW RESTORE.

Command `ws x0 y0 x1 y1 [index]`

Arguments: `x0 y0 x1` y1 – rectangular area to save

`index` – optional argument if more than one area is to be saved. Note that the storage areas overlap, which restricts the size of each saved area. See tables below

For example, if two areas are to be saved, use index 0 and 2 and observe the size restriction below. Note that the limit is the product of the screen area's width (x1 - x0 + 1) and its height (y1 - y0 + 1). As long as that product is less than the max value shown for the target save area, it will fit.

8 bit color (palletized)		
One area	**Two areas**	**Four areas**
Index none or 0; max WxH=108544	Index 0; max WxH=54272	Index 0; max WxH=27136
		Index 1; max WxH=27136
	Index 2; max WxH=54272	Index 2; max WxH=27136
		Index 0; max WxH=27136

high color		
One area	**Two areas**	**Four areas**
Index none or 0; max WxH=54272	Index 0; max WxH=27136	Index 0; max WxH=13568
		Index 1; max WxH=13568
	Index 2; max WxH=27136	Index 2; max WxH=13568
		Index 3; max WxH=13568

Example: `ws 0 180 160 239`

Saves the lower left eighth of the screen to index area 0.

WINDOW RESTORE (not available for SLCD or SLCD43)

Description: Restores previously saved rectangular screen area.

Command `wr x y [index]`

Arguments: `x y` – top left corner of area

`index` – optional argument; see WINDOW SAVE.

WINDOW RESTORE RECTANGLE (not available for SLCD or SLCD43)

Description: Restores previously saved rectangular screen area saved with the binary download command.

Command `wrr x y <width> <height> <index> [<offset>]`

Arguments: `x y` – top left corner of area

`Width` – width of image

`height`– height of image

`index` – a number between 0 and 3 referring to the portion of memory to start retrieving data from.

`offset` – optional argument. Offset into off-screen memory to start retrieving pixel data. The default offset is 0.

Notes:

1. If the offset points somewhere other than the beginning of the image data, the beginning or last pixels in the image may display data outside the range of the stored image (See BINARY DOWNLOAD).
2. Command not supported on “SCLD” controller boards.

BINARY DOWNLOAD

Description: Enables a raw binary data stream to be written to the SLCD flash memory or frame buffer. This is used by BMPload to update the stored bitmaps and macros.

Command `bdld <index> <offset> <size> <timeout>`

Arguments:

`index` – a number between 0 and 5 referring to the type and location of memory to store data. Indices 0 - 3 refer to the four areas used by the "window restore" command. Index 4 refers to on flash memory. Index 5 refers to on-screen memory.

`offset` – offset from selected memory area to store pixel data

`size` – number of bytes to store in memory

`timeout` – maximum delay in milliseconds between bursts of data from the host computer. If the host computer fails to respond within this period, an exclamation is returned and the binary download terminates.

Notes:

1. If the command is accepted, the SLCD issues a standard 2 byte prompt '>',0x0d. From then on all received data is handled as binary. On successful completion, another standard prompt is issued. If there is a timeout, a 2 character error prompt '!',0x0d is issued.

2. Software flow control from the SLCD to the host *MUST* be obeyed. No more than 64 characters may be sent after an XOFF (0x13) is received by the host.

3. The SLCD Flash memory must be erased before bdld can be used to update its contents. The erase command is “xmc 0xFEEB”.

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 BMPload PROGRAM.

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 MACRO FILES AND FORMAT.

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

`m <macro_name> [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.

Note: The maximum number of arguments is 10, and maximum size of each argument is 8 characters.

Examples: `m2`

This causes macro #2 to execute.

`m abc_xyz "" 2`

This causes the macro named "`abc_xyz`" 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`

MACRO ABORT

Description: This command stops execution of the current running macro. In addition, the command flushes (resets) the incoming command buffer.

Command: `*abt`

Host Notification: '>'<return>

The success prompt, indicates successful abort of executing macro.

Example: `*abt`

The example above could be a case when the host has sent a bunch of commands to update data on a screen. While those commands are being processed by the SLCD, a button is pushed which means "draw different screen". The host then sends a MACRO ABORT command and waits for the response. The SLCD can start drawing the new screen immediately, instead of having to wait until the previously buffered commands were finished.

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 | name >> [<macro2 index | name >]`

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 a 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.

<name> is the macro name of a macro which has an index. This can be used as an alternative to <macro index>.

Within the <name> an optional pre-defined label (see Labels section) may be concatenated. This label format only applies to labels used with this command. The specific format of the predefined label is related to the response for the type of hotspot or button. The format for the predefined label is:

':'<response character: 'x' | 'r' | 's'><touch index>[_<state: '0' | '1'>]

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"

bdc 1 100 100 20 "TURN ON" "TURN OFF"

xm 1 button_laction:s1_1 button_laction:s1_0

This creates a latching button with centered text. The initial state is 0. When the button is pressed and the state becomes 1, macro "button_laction" is executed, including the macro statements following the label "s1_1".

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 | name> [<macro2 index | name>]`

Arguments: See TOUCH MACRO ASSIGN.

Example 1: `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.

Example 2: `xmq 5 draw_macro`

This will run macro name “draw_macro” 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.

Command: `xa[q] <n> action <index | name><args>`

Arguments: `<n>` 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.

`<index>` 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.

<name> the macro name of a macro which has an index. This can be used as an alternative to <macro index>.

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

Example 1:

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

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

```
#define test 17
t "`0`" 10 20
#end
```

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

```
t "Check" 10 20
```

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
```

ANIMATION DEFINE

Description: Defines a sequence of commands to be played back continuously or on demand, concurrently and independent of commands received from the communications port, macros and buttons. A delay function (see "Yield") is used to suspend the animation for a specified period of milliseconds. The animation may be suspended at any "Yield" point (see "anid").

Animations are disabled when defined and must be activated using the `"anie"` (animate enable) command. An animation without a yield is executed once and suspended at the end of the animation. Animations cycle continuously unless a "yield stop" is contained in the animation script, the animation lacks a yield, or the "anid" command is issued to stop the animation.

Command: `ani <n> <text string>`

Arguments:

`<n>`	The index of the animation, 0 through 9
<text string>	Any valid command *Except* animation or flashing text.

Note: To control an animation using an animation script, the controlled animation or flashing text must be defined before defining the controlling animation. Only one animation may be defined at a time. Each "ani" command is used to define one graphics command; multiple commands may be incorporated into a single animation by breaking the display list into multiple lines, one command per line. Defining an animation with a different index closes the previous animation. Use the "anie" and "anid" commands to activate and deactivate defined animations. Use the "anic" and "anix" commands to delete animations.

Example: The list of commands below implements the "New Features" demo included with the kit. Comments were added to assist the reader and are stripped when received by the display.

```
xi 7 0 0                        // Background bitmap
anic                            // Clear animation

// setup font and color for TF command
f 24B
S 0f0 fff                       // Green text
tf 0 "FLASHING TEXT" 45 110 T

// Define animation #1 – Flashing LEDS
ani 1 xi 27 150 130             // Left LED On
ani 1 y 50                      // wait 50 MS
ani 1 xi 26 150 130             // Left LED Off
ani 1 xi 27 210 130             // Middle LED On
ani 1 y 50                      // Wait 50 MS
ani 1 xi 26 210 130             // Middle LED Off
ani 1 xi 27 270 130             // Right LED On
ani 1 y 50                      // Wait 50 MS
ani 1 xi 26 270 130             // Right LED Off
ani 1 y 50                      // Wait 50 MS
// End of animation #1
anie 1                          // Start animation 1

// Define animation #2 – “ROTATE” left continuously
ani 2 k 226 70 300 100 1 L      // “ROTATE” left
ani 2 y 50                      // Wait 50 MS
// End of animation #2
anie 2                          // Start animation 2
k 100 60 180 110 10 u           // Move “scroll” up

// Define animation #3 – “SCROLL” moves Down
ani 3 s 0 1                     // Scroll fill color
ani 3 k 100 60 180 110 1 d      // Scroll Down
ani 3 y 50                      // Wait 50 MS
// End of animation #3

// Define animation #4 – “SCROLL” moves Up
ani 4 s 0 1                     // Scroll fill color
ani 4 k 100 60 180 110 1 u      // Scroll Up
ani 4 y 50                      // Wait 50 MS
// End of animation #4

// Define Controlling animation #5, This animation
// selectively enables and disables animation scripts
// 3 and 4 successively for a period of ½ second.
// The effect is “SCROLL” scrolls up for a period
// ½ second, down for ½ second and repeats.

ani 5 anie 3                    // Enable Scroll Down
ani 5 y 500                     // wait ½ Sec.
ani 5 anid 3 0                  // Stop at first yield
ani 5 anie 4                    // Enable Scroll Up
ani 5 y 500                     // Wait for ½ Sec.
ani 5 anid 4 0                  // Stop at first yield
// end of animation #5
anie 5                          // Start animation #5
S 000 fff
```

ANIMATION LIST

Description: Lists the animation to the serial port for editing or incorporation into a macro, button or script. The animation is listed in a form suitable for cut and paste into other scripts. This method can be used to develop and tune animations, then incorporate the completed animation into a script.

Command: `ani? <n>`

Arguments: `<n>` The index of the animation, 0 through 9. If no parameter is given, the amount of free animation space in bytes is returned.

Example: When the command `tf 0 “hello world”` is entered to create a text flash animation.

ani? 0

Returns:

```
ani 0 f 13B
ani 0 S 000 fff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 t "Hello world"
ani 0 y 500
ani 0 f 13B
ani 0 S fff 000
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm T
ani 0 t "Hello world"
ani 0 y 500
```

ANIMATION YIELD

Description: Suspends (sleeps) an animation for <Milliseconds> or stops the animation.

Command: `y [<Milliseconds> | stop]`

Arguments: <Milliseconds> Number of milliseconds to sleep this animation.

stop Halt this animation until ANIE command issued.

Note: The Yield command is only valid when executed in an animation script.

ANIMATION DISABLE

Description:	Stops the animation specified by animation index and yield #.
Command:	`anid <n> <Yield #>`
Arguments:	<n> The index of the animation, 0 through 9
	<Yield #> A reference to one of a animation's yield commands.
	Yields are numbered for each animation starting at 0 (zero) and continues up to the number of yields-1 contained in that animation.
Note:	Animations are "stopped" by advancing and executing those commands between the previous and selected yield.
Example:	anid 0 0
	Stops animation 0 at the first yield command.

ANIMATION ENABLE

Description:	Enables animation execution for a specified animation
Command:	`anie <n>`
Arguments:	<n> The index of the animation, 0-9
Example:	`anie 0`
	Enables animation 0

ANIMATION CLEAR

Description:	Clear the animation and flashing text definitions and disables the animation engine.
Command:	`anic`
Arguments:	None
Example:	`anic`
	Clears the animation buffers and stops the animation engine.

ANIMATION DELETE

Description:	Deletes the selected animation script.
Command:	`anix <N>`
Arguments:	`<n>` The index of the animation, 0 through 9
Example:	`anix 0` Removes animation 0, reclaims animation memory.

ANIMATION SYNCH

Description:	Restarts all animations at beginning of scripts.
Command:	`anis`
Arguments:	`None`
Note:	For best results, stop the animations using the anid command with the proper yield points to prevent graphics residue. Possible uses of this command are synchronizing flashing text or lamps.
Example:	`anis` Any running animations are restarted.

WAIT VERTICAL RETRACE

Description:	Returns when a vertical display retrace occurs with an optional offset. Used to avoid "tearing" in animations.	
Command:	`wvr [<line>] [<line2>]`	
Arguments:	`<line>`	Optional number of vertical lines to wait after retrace. Used to wait until the display trace has passed a specified vertical display line.
	`<line2>`	Optional value added to first argument for total line offset. Useful when displaying a bitmap that starts at line and is line2 high.
Note:	For best results, bitmaps for animations should be stored uncompressed in flash memory.	
Example:	`wvr 100 35` This waits until vertical refresh plus 135 vertical lines. Useful to put in an animation script before displaying a bitmap a x,100 with height 35.	

OUTPUT STRING (MAIN)

Description: This outputs a text string to the main 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:

`\\`	single backslash
`\"`	double quote
`\n`	line feed
`\r`	return
`\xhh`	arbitrary character with hex value hh
`L<idx>`	replaced with value of slider <idx>
``	single backtick

Example:

```
out "\x48ello \"world\"\r"
```

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

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

OUTPUT STRING (AUX) - SLCD compatible

Description: Same as OUTPUT STRING (MAIN), except that the string is sent to the "aux" port. Argument rules are the same, except that a null byte may be sent if it is the first byte in the string. The aux port is COM1 if the main port is COM0, and it is COM0 if the main port is COM1. This command is compatible with the SLCD version.

Command: `aout "<text string>"`

WRITE TO AUX PORT

Description: Writes string to specified communications port. Note that whatever port is acting as the main port cannot be written to this way; use the OUTPUT command. For example, if the main port is 2, then the aout2 command will return an error ("!"). Standard hex escapes are supported. A null byte must be sent at the start of a string or by itself.

Command: `aout<0-3> "<your message>"`

Argument: Port as shown above

Quoted string to send "<your message>"

Example: `aout0 "hello world\x01\xff"`

Sends "hello world" followed by two bytes hex 01 and hex FF to the serial port COM0. If this command is entered from COM0, it will fail.

Example: `aout3 "\x00"`

Sends a single byte 0x00 to the USB port.

READ FROM AUX PORT

Description: Reads serial data from specified AUX port. The AUX port receive buffers are 80 bytes long (79 characters plus a NULL char). If the buffer becomes full, any further data is thrown away. An escape sequence is used to receive null bytes: nulls are translated into the string "`\0`" and the '\' character is translated into "\\".

Command: `ain[0|1|2|3]`

Argument: Port as shown above

Example: `ain3`

Returns: `:<received data from COM 3><prompt>`

Note: A colon is pre-pended to message, and the messge is terminated by the standard 2 byte prompt 0x3e 0x0d ("><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 SLCDexternal flash memory. See Appendix D for details.

Note that this same effect can be performed using a power-on macro. The splash screen capability was provided for users who do not have macros.

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, so this command only needs to be executed once.

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, so this command only needs to be executed once. 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 "bv" command stores the value in non-volatile memory and it is restored on power-on. The "bvs" command effects a change that is not maintained over reset. It is much faster to execute.

Command: `bv [+|-]`<level> Set beep volume, and make it "permanent"

Command: `bvs [+|-]`<level> Temporary volume change

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.

If no arguments are provided, the current level is returned.

BEEP FREQUENCY

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.

Note: The beep frequency is set at factory to generate maximum loudness level.

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 from 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 unless the optional 's' is added to the basic "xbb" command.
Command:	`xbb[s] [+|-]<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.
Example:	`xbbs 128`
	This will set the brightness to half the maximum value, and it is temporary (value not restored at power-on).

SET BAUD RATE

Description	Sets a baud rate of COM 0-3 serial port. This is temporary and the unit will revert to the default setting the next time power is cycled.

Command:	`baud [230400|115200|57600|38400|19200|9600]`
	or
	`baud 0-3 [230400|115200|57600|38400|19200|9600]`

Argument:	baudrate as shown above
Note:	USB port (COM 3) can be set to 460800 baud.
Example:	`baud 57600`

VERSION

Description:	Displays the version and configuration of the firmware.
Command:	`vers`
Returns:	Version <version number<matrix addressing>>/<color depth>/<orientation> <board model> <"<panel string>">
	Version number - <major>.<minor>.<build> (all values numeric) Matrix addressing - "tft" (active) or "c" (passive). Color depth – "hc" (High Color) or "8" (8-bit color) Orientation - "L" (landscape) or "P" (portrait) Board model – "SLCD6", "SLCD43", "SLCD" Panel string – Contact REACH Technology with Panel String to determine display panel information

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`

SET LED

Description: Turns the LED D2 on the board on or off. 1 is on and 0 is off.

Command: `led [0|1]`

Returns: > (standard prompt)

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: Returns 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 (or a section of it) used to store macros and bitmaps. This can be used in production code to verify that the correct bitmaps are loaded in the board. *With no arguments, returns the CRC of the whole flash (from 0x0 to 0x2FFFFF).*
Command:	`*CEXT [<from> <to>]`
Arguments:	`[<from> <to>]` optional start and stop offsets used to specify a section of the external flash to CRC. Valid ranges are 0-0x2FFFFF.
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 the factory manufactured state with the exception that the contents of the external flash memory (bitmap and macro storage) is not affected. Note his does not reset the beep frequency to the manufactured state which was calibrated for maxium volume (resonance).
Command:	`*MFGRESET`
Returns:	"Power on" prompt.

DEBUG TOUCH

Description:	Used for Reach internal debugging; serial output with debug on 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 COMMAND

Description: Used to enable command debugging. When the command is set to one, the failing interpreted command is displayed. Commands present on the command line and macro (if applicable) are displayed.

Command: `*cmddebug <0|1>`

Returns: <on | off> <*return*>

Command line failure: :cmd err "<*failing command line contents*>"

Macro failure: :macro #<*macro number*> [(*macro name*)] err "<*failing macro line contents*>"

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 | name> [<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.

<name> = Text string of macro name. This can be used instead of index to set the power-on macro.

`<option>` = optional argument; 0 means display the power-on copyright, and 1 means do not display it.

Examples:

```
*PONMAC 2
*PONMAC Splash_macro
```

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>`
`r<index><return>`	`R<binary index>`
`s<index><state><return>`	`S<binary index><binary state>`
`m<index><return>`	`M<binary index>`
`e<index><return>`	`E<binary index>`
`<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. This command's argument is the a macro index or name.

Command: `*DEMOMAC <index | name>`

SET VARIABLE

Description: Used to set a value to an internal variable. If an internal variable (Integer, String, and Point Coordinate) is used, it should be after this command to have a meaningful value. See Special Arguments section for variable details.

Command: `set <internal variable name> <value>`

Arguments: `<internal variable name>`

`Integer – i0 thru i9`

`String – s0 thru s9`

`Point Coordinate - p0 thru p9`

Example: `set i9 -200`

Set internal integer variable i9 to negative 200.

`Set s5 “Hello World!”`

Set internal string variable s5 to the string value, “Hello World”.

GET VARIABLE

Description: Used to return the value of an internal variable (Integer, String, and Point Coordinate). See Special Arguments section for variable details.

Command: `get <internal variable name>`

Arguments: `<internal variable name>`

`Integer – i0 thru i9`

`String – s0 thru s9`

`Point Coordinate - p0 thru p9`

`Returns <value><return>`

Example: `get i9`

`-200`

Internal Integer variable i9 returns its value, negative 200.

`get p5`

`20 200`

Internal Point Coordinate variable p5 returns its value, x coordinate, 20 and y coordinate, 200..

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 loaded into firmware.

Command: `*panel`

CONTROL PORT AUTOSWITCH

Description: The SLCD has two and the SLCD6/43 have four serial ports. Only one port is active at a time as the unit's control port. In certain circumstances it is useful to be able to switch which port acts as the Main port temporarily.

Command: This can be done by sending three consecutive <return> characters to the port that is to become the new main port. Once this is done, the Main and Aux ports will swap. Note that a different escape character than <return> can be used – see AUX ESCAPE below.

SET AUX ESCAPE

Description: Used to set the escape character for the control port autoswitch. Saved in non-volatile memory; this command only needs to be executed once.

Command: `*auxEsc <hex value of ASCII character>`

Example `*auxEsc 1b`

This sets the escape character to the ASCII Esc code

SET CONTROL PORT

Description: Used to set the port used to control the unit. This is stored in non-volitile memory and will be used on power-up; this command needs to be executed only once. It also sets the "aux" port to either COM0 or COM1, whichever is not the main port.

SLCD Command: `*com<0-1>main`

SLCD6/43 Command: `*com<0-3>main`

SLCD6/43 Example: *com3main

USB serial port is used as the main console.

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`

EEPROM READ / WRITE

Description:	Used to read and write from the non-volatile memory. Only 16 locations are user-writable. The return value is a two character ASCII hex value with the letters A-F in caps.
Command:	`*eer <hex location>` `*eew <hex location> <hex value>`
Arguments:	`<hex location>` is a single character in the set 0,1,..9,a,b..f `<hex value>` is one or two characters in the set 0,1,..9,a,b..f
Example1:	`*eew 2 a5`
Example2:	`*eer 2`
Returns:	A5`<return>`

DISPLAY OEM BITMAP IMAGE

Description:	Copies factory programmed bitmap onto the screen at x y (top left corner of bitmap target). Returns syntax error if bitmap is not defined.
Command:	`i <number> x y`
Arguments:	`<number>` is bitmap number:
Note:	These bitmaps are OEM defined, stored in the microcontroller code flash memory and are not downloadable. Contact Reach to have these installed.
Example	`i 1 0 0` This displays the first bitmap on the screen

COLOR TEST

Description:	Displays all possible 4096 colors in a timed sequence. Used to detect panel data cable opens or shorts. This is needed because in the 8 bit palletized color mode, a single image cannot display all possible colors.
Command:	`*TESTC`

SET ORIENTATION (rotate display 180 degrees)

Description: NOTE THIS IS PANEL-FIRMWARE DEPENDENT. NOT ALL PANELS SUPPORT THIS FEATURE.

Some panels have a hardware capability to rotate the display 180 degrees. For those panels, this command provides the ability to rotate the display on the fly. The touch calibration is flipped as well.

Command: `*orient [0|1]`

Argument:

`0`	normal orientation
`1`	flipped orientation

CHECK COLOR MODE

Description: Check if the controller firmware is 8 bit or high color. Backward compatible with early SLCD versions which do not have this command implemented.

Command: `*IS16`

Returns:

`'!'<retuurn>`	8 bit color
`'>'<return>`	high color

3. BMPload PROGRAM

3.1. *Overview*

The SLCD contains flash memory used for storing bitmaps, macros, and fonts. Stored bitmaps are displayed on the screen using the "xi" command and are used in creating objects such as buttons, meters, and sliders. The BMPload.exe program generates a load image containing the bitmaps, an optional macro file, and optional font files. This image is then either stored in a file or loaded into the SLCD/6flash memory. Once downloaded, the image is non-volatile; that is the contents are permanently stored even if power is off.

The download process clears the entire flash memory.

The SLCD6/43 can operate in 8 bit palletized color mode, or in high color mode. The selection is made by downloading the appropriate firmware into the controller. The high color firmware has "_hc" in the file name.

3.2. *8 Bit Color Mode Bitmap Format*

These bitmaps are known as 8 bit indexed color. This is also known as 256 color or palletized color mode. Bitmaps can be created with programs such as the Windows PAINT program, Adobe PhotoShop, or the Open Source editor GIMP. The PhotoShop palette file ps8666.act contains the palette used on the SLCD/6. Bitmaps that use this palette take less storage and display faster than ones that have an arbitrary palette.

As an alternative, a custom palette can be used, but this must be the same for all images.

3.3. *High Color Mode Bitmap Format*

If the SLCD is running high color firmware, BMPload will accept 1, 4, 8, or 24 bit color bitmaps. The BMPload program does a conversion between 24 bit file and 16 bit internal storage format. The 16 bit format is 565 - 5 bits for red and blue, and 6 bits for green. *Note: some graphics programs can store a .bmp file in 16 bit format. There is no standard for this format and it is not supported.*

3.4. *Bitmap compression*

The BMPload program can compress bitmaps using the RLE (Run Length Encoding) method. This is very efficient space-wise for control surfaces that have horizontal lines of constant color. However they are slower to display. Small images are not compressed as they do not take up much space. To disable compression of larger images, insert the string ".unc" into the file name, e.g. "01_MyBitmap.unc.bmp". This tells the BMPload program not to compress this file's image. Background bitmaps for slider objects and meter objects and sliding graphics ("xio" command) *need to be uncompressed*.

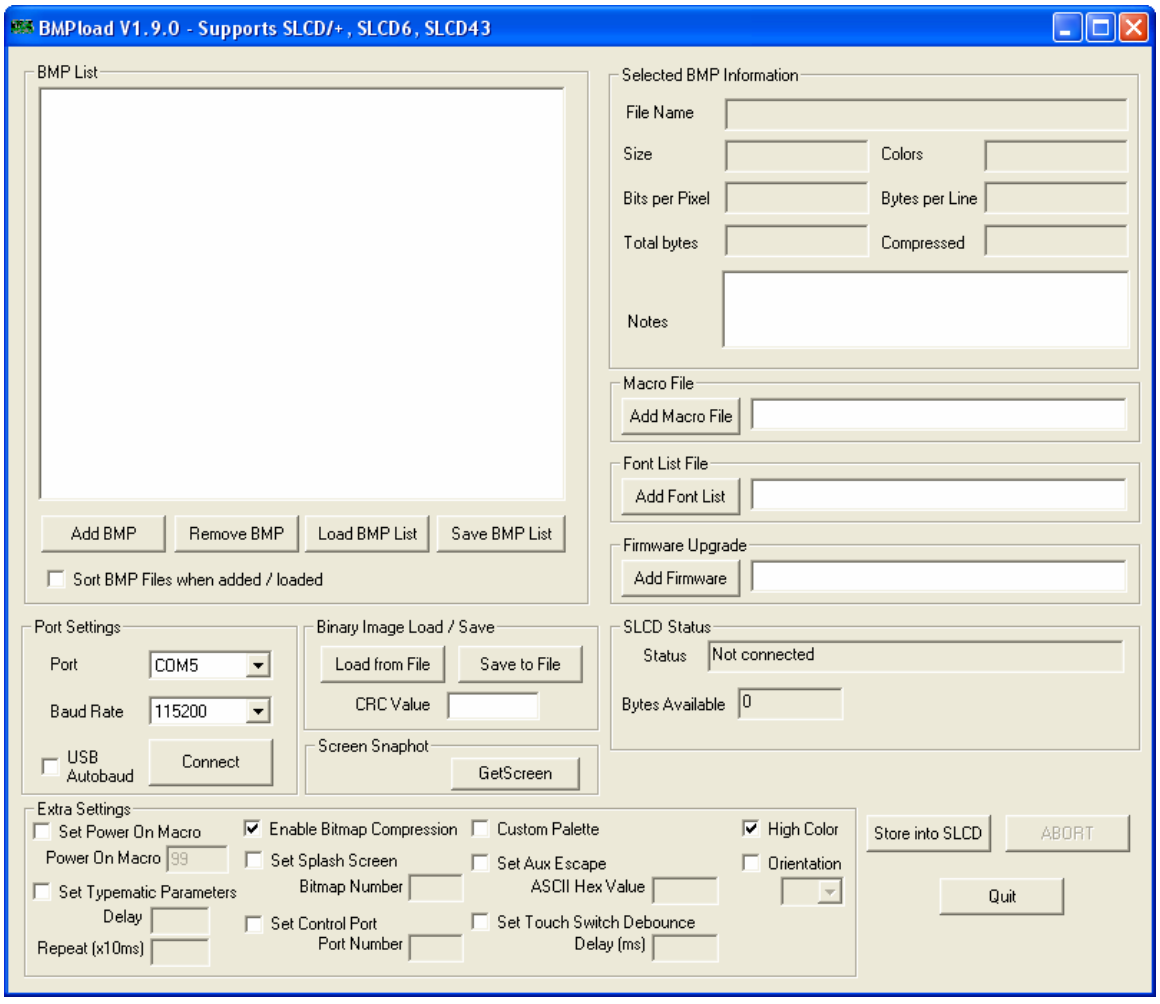
3.5. *Bitmap file naming convention*

Bitmaps are referred to by index number. In order to keep the index and bitmap in sync, the files should have the index number pre-pended to the file name. This way when the list is sorted alphabetically the index will match the bitmap. So, for example, the first bitmap could be named "001_whatever_you_like.bmp", the second "002_MySecondBitmap.bmp", and so on.

3.6. *Program Operation*

BMPload runs under Windows 98 through Vista. In order to download bitmaps and so forth to the board, the computer running BMPload should have a serial port connected to the SLCD. It can be run without a board connected if the purpose is to generate a BIN file.

When first run, the program looks like this:



The buttons and checkboxes operate as follows:

Add BMP

Add one or more bitmaps to the BMP List window. The List is automatically sorted in alphabetical order. Make sure that no two bitmaps have the same numeric prefix.

Remove BMP

Remove one or more bitmaps from the BMP List window.

Load BMP List

Load a list file containing the names of the bitmap files, one per line. This is a simple text file.

Save BMP List

Save the bitmaps in the BMP List window to a text file.

Sort BMP files when added / loaded

This option allows files to be sorted as they are added. This is useful when the files have sort-friendly prefixes such as 001_file1.bmp, 002_next.bmp, and so on.

Add Macro File

Selects a macro file (plain text file) to be incorporated into the load image.

Add Font List

A font list is a simple text file with each line containing a font name alias, and the name of the associated font file. These fonts will be included in the load image. *It should be noted that a font name alias is limited to 8 characters.*

Add Firmware

This is used to update the board firmware. In general, the firmware and bitmap / macro / font image can be loaded at the same time. Exceptions are when the color support is changed (8 to 16 or vice versa), or when the load image is larger than the flash memory size minus the firmware image size.

Port Settings - Port

Selects the COM port to communicate with the SLCD board.

Port Settings - Baud Rate

Selects the COM port baud rate to communicate with the SLCD board. Note that "standard" PC COM ports are limited to 115200 baud.

Port Settings - USB Autobaud

If the SLCD6 or SLCD43 controller's USB port is connected to the PC, checking this box will make the program attempt to change to the highest supported baud rate for the download.

Port Settings - Connect / Disconnect

If the Connect text is shown and pressed, a connection is attempted with the Port and Rate Settings as selected. If Disconnect is shown and pressed, the port is disconnected.

Binary Image Load / Save - Load from File

Loads a previously stored image file (see Save to File below). Also sets a flag to compare the CRC when finished programming.

Binary Image Load / Save - Save to File

Instead of storing the data directly into the SLCD over the serial line, this option saves the same data in a binary file to be used later with the "Load from File" button. This is typically used to prepare a production image or for In Application Programming. The state of the Extra Settings flags is also saved, as well as the CRC value. This makes it easier for production - only one file is needed for loading.

Binary Image Load / Save - CRC Value

This value is filled in when the SLCD image is programmed via the serial port. It is retrieved from the SLCD via the *CSUM command. If this value is manually entered by the user, or is set by the "Load from File" button, then BMPload will check this value against the retrieved board value after programming and generate an error if they are not equal. This aids in production programming.

Store into SLCD

This starts the serial download process. An SLCD must be attached to the specified serial port. The status is shown in the Status text box. When complete, a sound will play and the serial port automatically disconnects from the SLCD. This is helpful in the case where one PC serial port is used for both download and serial control (e.g. Hyperterminal).

Set Power On Macro

Use this to set a power-on macro as part of the load image. Implements the `*PONMAC` command.

Set Typematic Parameters

Use this to set typematic button parameters other than the default. Implements the `typematic` command.

Set Splash Screen

Use this to set a splash screen. Implements the `*SPL` command.

Set Control Port

Use this to change the default control port. Implements the `*com?main` command.

Set Aux Escape

Use this to change the aux escape character from the default 0x0d (return). Implements the `*auxEsc` command.

Set Touch Switch Debounce

Use this to change the touch debounce from the default 100ms. Implements the `*debounce` command.

Enable Bitmap Compression

This box is normally checked which means that all bitmaps above a certain size are compressed. This checkbox allows all bitmaps to be stored uncompressed. The tradeoff is speed versus size: compressed are smaller but slower to display. Independent of this checkbox, any file with the string ".unc" in the file name will not be compressed. To see if a file will be compressed or not, highlight the file name in the "BMP List" window and the information will be displayed to the right.

Custom Palette

This is only applicable to 8 bit color firmware. The SLCD has a standard internal palette. Bitmaps with this palette display faster, and all bitmaps have their palette mapped to this one. If the standard palette is not ideal, then all bitmaps can have a custom palette, and this box needs to be checked. The custom palette is loaded into the hardware on power-on.

High color

This box needs to be selected if the SLCD6 is running 16 bit color firmware. For SLCD or SLCD6 8 bit color firmware it should be unchecked. This option is provided so that a bitmap binary can be saved without having an SLCD connected.

Orientation

This control consists of a checkbox, and then a drop-down menu is enabled for options. See the command “SET ORIENTATION (rotate display 180 degrees)” for details.

3.7. *Bitmap order*

The order of the bitmaps in the BMP List window is important because the DISPLAY DOWNLOADED BITMAP IMAGE command uses the bitmap index, which is simply the line position of the bitmap in this window. In other words, if the list showed:

abitmap.bmp
another.bmp
last.bmp

then the command to display bitmap 2 at x = 0, y = 0, "`xi 2 0 0`" would display "another.bmp".

The best way to keep this clear is to start the bmp file name with its index number, for example "001_first_bitmap.bmp", "002_second_bitmap.bmp", and so on. By using three digits, up to 999 bitmaps can be alphebetized.

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.

The easiest way to organize bitmaps is by using a BMP List file. This is an ASCII text file that simply contains a list of bitmaps on each line. See the example "demo.lst" file on the kit CD in the "BMPs and Macros" folder. It is recommended that the bitmaps have their order in the file name, e.g. "01_first bitmap.bmp". In this way, it is easy to keep them in order and to remember the required index number for the "xi" command.

3.8. *CRC Check (Production)*

In a production setting, it is useful to verify that the download has been completed accurately. The best way to do this is to store the production image in a file ("Save to File") and then use "Load from File" in production. This will load all the saved checkbox settings AND the CRC. If the CRC is loaded this way, or by hand, when the download is complete, the CRC of the data flash will be checked against this value and an error message generated if the CRC is wrong.

The operation of the CRC Value is as follows:

1. If the box is empty when "Store into SLCD" is clicked, it will be filled in with the CRC reported by the SLCD after the download is finished.
2. If the box is filled in by the used or by the "Load from File" button, then its value will be compared with the CRC reported by the SLCD after programming and an error generated if they are not the same. In this case, the box will NOT be updated with the reported value as it assumed that the failed compare indicates that a programming error occurred.

3.9. *BMPLoad speed issues*

The BMPload program will work with most PC serial ports. The standard PC serial ports only support a maximum of 115200 baud. The SLCD serial ports can be set to 230400 baud and on the SLCD6 and SLCD43 the USB port to 460800 baud.

Recommended USB serial port adapters are those with Prolific or FTDI chips. See http://www.ftdichip.com

3.10. *Custom palette (8 bit color only)*

The SLCD standard palette provides 16 shades of gray plus 6 shades of each color. This is what is known as a uniform palette. For a specific “look and feel”, it may be desirable to use a custom palette. To do this, all bitmaps must be created using the same palette of 256 colors. Then, when the BMPload program is used, check the “Custom Palette” option box. When the SLCD powers-on, it will load the custom palette.

Notes:

1. With a custom palette, the SET COLOR command takes palette index values as arguments, not specific colors, since the color-to-index mapping is not known.
2. With the Custom Palette selected, after the BMPload program finishes, the new palette is loaded and the screen may change color. This is due to the palette change.

The Adobe Photoshop program is well suited to generating bitmaps with a specific color palette.

4. MACRO FILES AND FORMAT

4.1. *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 (arguments) 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 NUMBER OF MACROS = 254
- MAXIMUM CALL DEPTH = 4

  A macro can call another macro, but only to a depth of 4.
- MAXIMUM ARGUMENTS PER MACRO = 10
- MAXIMUM CHARACTERS PER ARGUMENT = 8
- MAXIMUM TOTAL STORED ARGUMENTS = 50

  (stored via the TOUCH MACRO ASSIGN WITH ARGUMENTS command)

4.2. *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 macro definition file can be used to load the macros into the SLCD flash memory. There are two versions to choose from when designing a macro file. The original version, version 1, takes two arguments <text_name> and <number>. This version requires that all macros be listed in numerical order starting at 1 and incrementing by 1. It has the disadvantage that editing a macro file can be cumbersome because you have to keep track of macro numbers.

Version 2 takes only the < text_name> argument. When using version 2 each macro definition is assigned a number based on the order in which it appears, starting with 1. This way, when using functions that refer to macros, the <text_name> can be used to reference them. When calling a macro in version 2 by the macro's name, you must include a space after the function name.

In BMPload version 1.7 or higher, every time a macro file is stored, a header file is created in the same folder with the same name as the macro file but with extension ‘.h’. These header files list all the macro defines and display every macro name with its assigned number. This header file can be used as a ‘C’ include file in the user’s microcontroller program.

The format for each macro in version 1 is as follows:

#define <text_name> <number>

(one or more command lines)

.

.

#end

The format for each macro in version 2 is as follows:

#define <text_name>

(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. In version 2 the name can also be used instead of the macro number when using a function that references a macro. All macro names must start with an alphabetical letter or an underscore but thereafter can also contain numbers.

In version 1 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.

Version 2 referencing example:

```
#define example_a       //assigned macro number: 1
m example_b             //can reference macro #2 by its name. Or…
m 2                     //can also reference macro #2 by its assigned macro
                        // number  (the order in which it appears)
#end

#define example_b       // assigned macro number: 2
*PONMAC example_a
#end
```

Also with BMPload version 1.7 or higher it is acceptable to indent lines:

```
#define example_a
        //indented lines are ok
        m example_b
        m 2
#end

#define example_b
        *PONMAC example_a
#end
```

4.3. *Macro Parameters (Arguments)*

Macros can be parameterized by using the special escape sequences `` `0` thru `9` `` in the command lines. These are replaced at execution time by the arguments supplied by the command that invoked the macro. The combined total length of all macro arguments for a macro call is 128 characters (command line length) minus the character length of the macro name or number plus spaces, and delimiters (ex. double quotes). Note the special escape sequence delimiter character ‘`’ has the ASCII value 96 decimal, 60 hexadecimal.

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
```

4.4. *Assigning macros to buttons*

The Touch Macro Assign command and variants can be used to automatically run a macro when a button is pressed or released. Note that doing this will cause any currently executing macro to quit running.

4.5. *Special macro arguments and commands*

Memory Commands

Memory commands were added to implement the keyboard in the demo macros that come installed with the SLCD kits. These allow a character string to be saved and manipulated. The character string is accessed as a special macro parameter.

The commands are:

`mpush [<string index>] "<string>"`

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

Example:
>set s0 "hello "
>mpush 0 "there"
>get s0
hello there
>

`mpop` [<string index>] <number of characters to pop>

This removes the <number> of characters specified from the end of the memory variable. If the argument is -1, then the memory variable is cleared.

Example:

```
>set s0 "hello 123"
>mpop 0 4
>get s0
hello
>
```

Special Arguments

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

Simple Math on integer arguments

```
`(`0`+ 1)`
```

This is replaced by the value of the macro's first argument plus 1. Simple math supports addition or subtraction, *but only on arguments with an integer value; ie: "abc" or "1.2" would cause an error.*

Memory variable

```
`M`
```

This is replaced by the string stored by the mpush command.

Integer variable

```
`i0` thru `i9`
```

This is replaced by a 32 bit signed integer. See the SET VARIABLE command.

String variable

```
`s0` thru `s9`
```

This is replaced by a character string with a maximum length of 80 characters. See the SET VARIABLE command.

Point Coordinate variable

```
`p0` thru `p9`
```

This is replaced by the string representing a coordinate of a point on the screen. The format is "<x coordinate value> <y coordinate value>". See the SET VARIABLE command.

Slider value

```
`L<index>`
```

This is replaced by the value of the slider defined by `<index>`.

Random number

`R<lo>:<hi>`

This is replaced by a random number in the range <lo> to <hi>.

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 that an escape character (hex 1B) followed by a <return> received from the serial port will halt a looping macro.

Labels

A special directive can be used to identify a location within a macro in order to selectively execute specific command lines when the macro is called with the optional label identifier. A label consists of a colon (':') followed by a maximum of 32 alphanumeric characters (label name). The first character of a label name must not be numeric. The line placement of the label must begin in column 1 of the line (colon in column 1). Here is the format of a label:

:<label name>

Example:

:attach

Labels are invoked by calling the macro in the normal fashion, but including the colon and label name directly after the macro number in a macro call. For example say when want to invoke label "attach" in macro number 8. The command would be:

m 8:attach

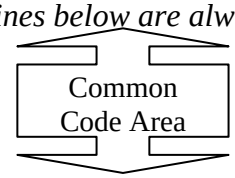
The format of a macro label invocation is: "m <macro name or number>:<label name>".

The execution flow of a macro invoked with an optional label starts with the "common code area". The common code area is a new feature with labels. The common code area is all command lines in a macro after the macro definition line (#define) up to the first label. So, first the common code area command lines are executed, then execution starts with the line after the matching label, and ends with the next label. Below are examples of a macro that uses labels. Command lines executed are in **bold**.

Example of Macro call with label

m 8:attach *(user calls macro number 8 with label ":attach")*

#define create_button //*(command lines below are always executed)*
S 333 CCC
f 24
t "Calling create_button" 200 10



:attach //*(command lines below this matching label are processed)*
t "Attaching button 1 to macro" 200 30
xa 1 p 3 0

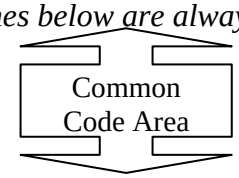
:define //*(execution stops here)*
t "Defining button" 200 50
bd 1 0 32 1 "INCREASE" 9 5 10 11

#end

Example of Macro call without a label

m 8 *(user calls macro number 8 without any label)*

#define create_button // *(command lines below are always executed)*
S 333 CCC
f 24
t "Calling create_button" 200 10



:attach //*(command lines below are not executed since there is no matching label)*
t "Attaching button 1 to macro" 200 30
xa 1 p 3 0

:define //*(command lines below are not executed since there is no matching label)*
t "Defining button" 200 50
bd 1 0 32 1 "INCREASE" 9 5 10 11

#end

Error conditions for a label include:

- label name is not found
- use of a label that results in no commands line being executed.

In both these cases an error message is transmitted to the user.

4.6. *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 1<return>
```

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

4.7. *Special Macro Usage Notes*

1. If a state button has a macro assigned and the button state is changed using the SET STATE BUTTON command, the associated macro will be run just as if the button has been pressed or released. This will also stop any running macro. To prevent this, disable the button, change the state, and then re-enable the button.
2. There are some conditions that can cause a bad power-on macro to prevent the board from communicating over the main console port. If this happens, send a continuous stream of <esc><return> (0x1b,0x0d) bytes while the unit powers up. This will prevent the power-on macro from running.

5. ANIMATION AND TEXT FLASH

5.1. *Introduction and limitations*

The animation feature allows the creation of command scripts that execute independently of, and are created and controlled by the macro, button or command stream. A total of ten animations may be created and executed at any one time (up to the limitation of available memory). Animations may be created or deleted by macros and the command stream. Animations cannot be created or deleted by other animations.

Animations can be stopped and started at select control points in the animation. These control points are called YIELDs. A YIELD suspends the animation and "yields" control to the next animation or the system. A yield normally specifies a time delay, with the animation stopping for <t> milliseconds before resuming or stopped with the "stop" parameter. The animation may be stopped at a specified yield point with the "anid" (animation disable) command. The "anid" parameters <index> and <yield> specify the animation index (from 0 to 9) and the "yield" in that animation. The yields in the animation are numbered from 0 to N-1. To stop animation "0" at the first yield, the command `"anid 0 0"` is used.

5.2. *Examples*

The text flash command "tf" uses animation to display flashing text. A text flash command, in expanded form as stored in the animation buffer is shown below. The font, foreground and background colors, text alignment, origin, and cursor position are taken from the current settings when the text flash command is issued.

The text flash command,

```
tf 0 300 "this is a test" 120 130 X
```

Using "power on defaults", the above command produces the animation script as dumped by the tf? command:

```
f 8x8
S fff 000
ta LT
o 0 0
sc 0 0
t "hello"
y 500
f 8x8
S 000 fff
ta LT
o 0 0
sc 0 0
tm T
t "hello"
y 500
```

6. FONTS

The SLCD and SLCD6 have built-in fonts as described in the following sections. The BMPload program can be used to download other fonts including Unicode fonts.

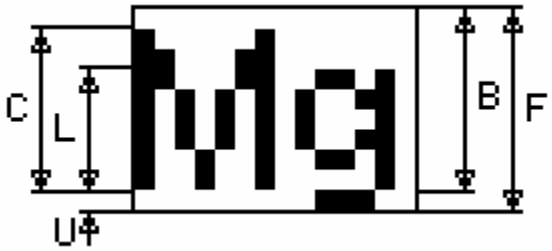
6.1. *Proportional Fonts*

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

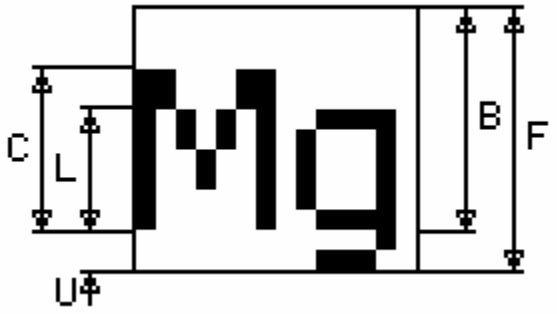


Font 10S – ISO 8859-1 (Latin1 or Western European)

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

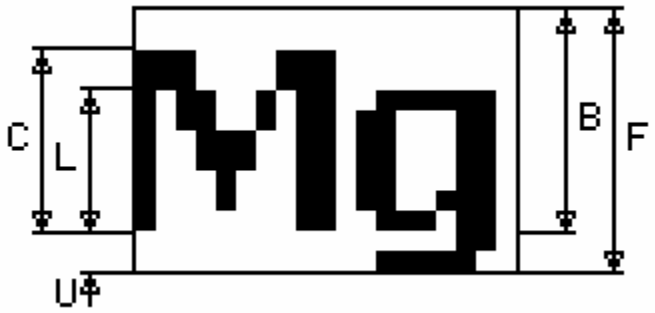
Font 13 – ISO 8859-1 (Latin1 or Western European)

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



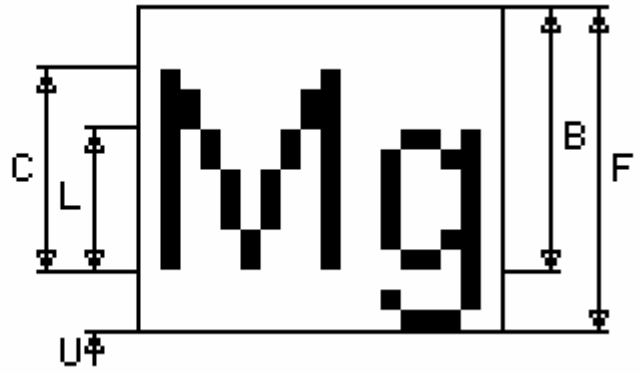
Font 13B – ISO 8859-1 (Latin1 or Western European)

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

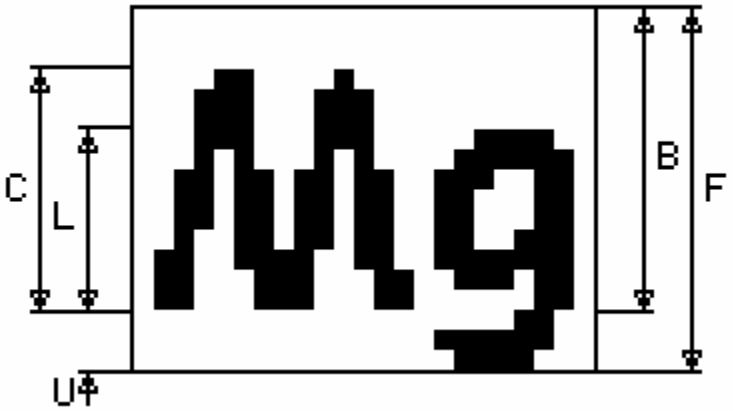


Font 16B – ISO 8859-1 (Latin1 or Western European)

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

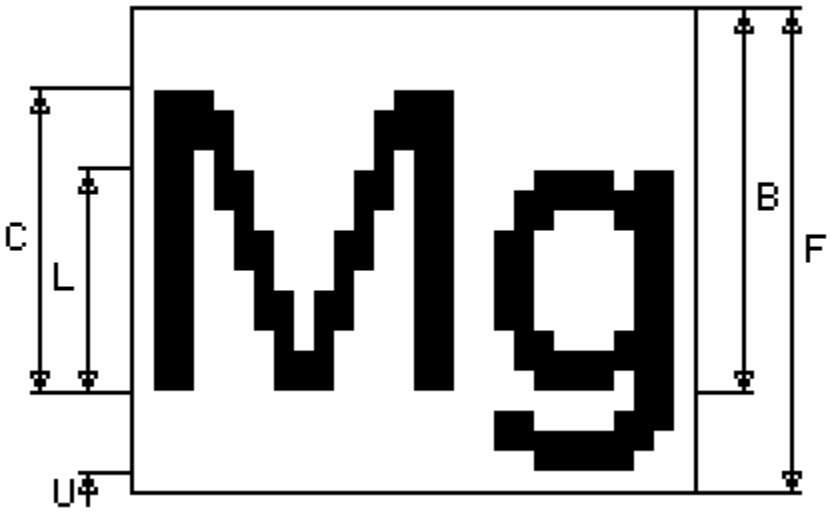
Font 18BC – ISO 8859-1 (Latin1 or Western European)

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

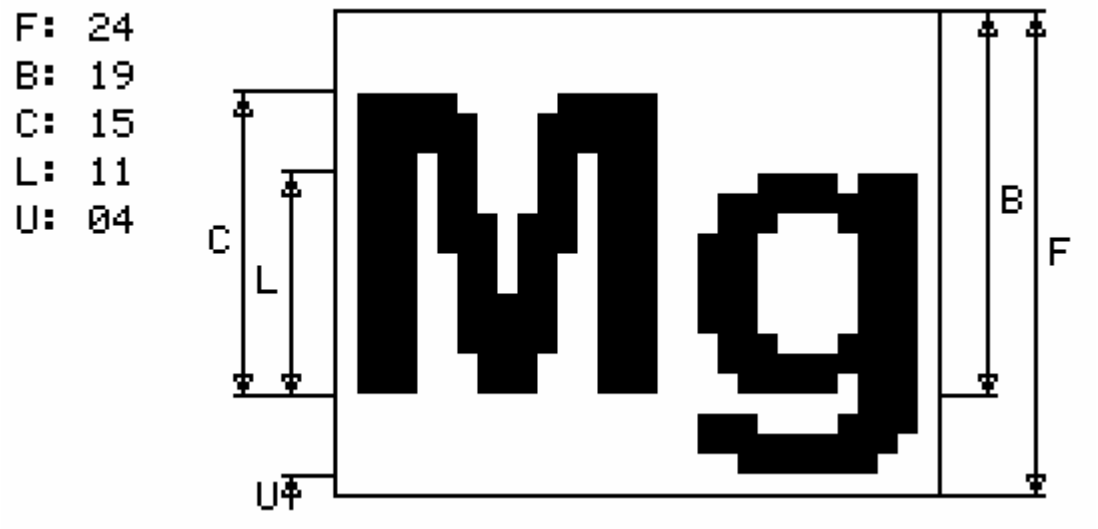


Font 24 – ISO 8859-1 (Latin1 or Western European)

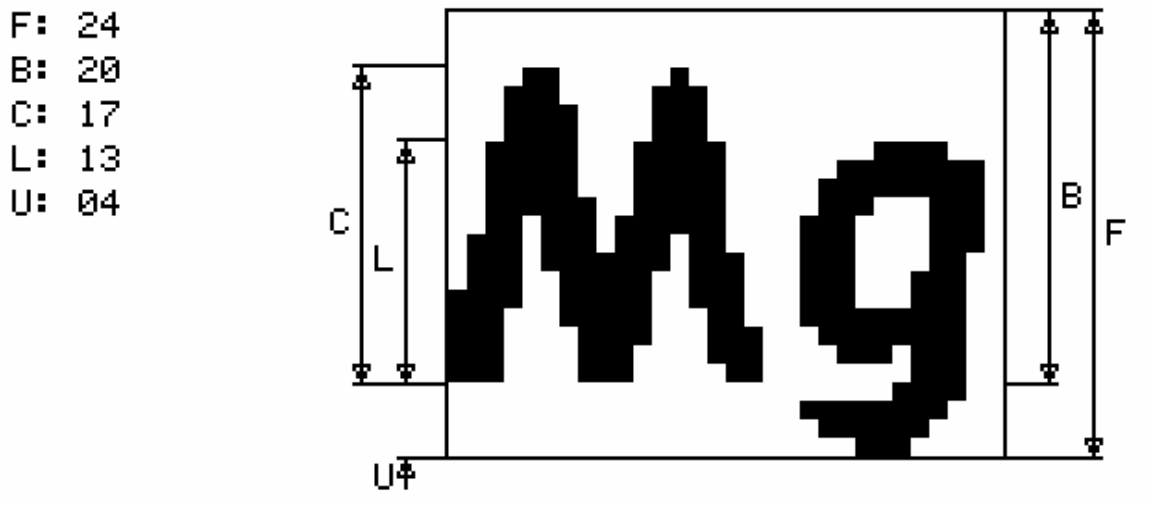
F: 24
B: 19
C: 15
L: 11
U: 04



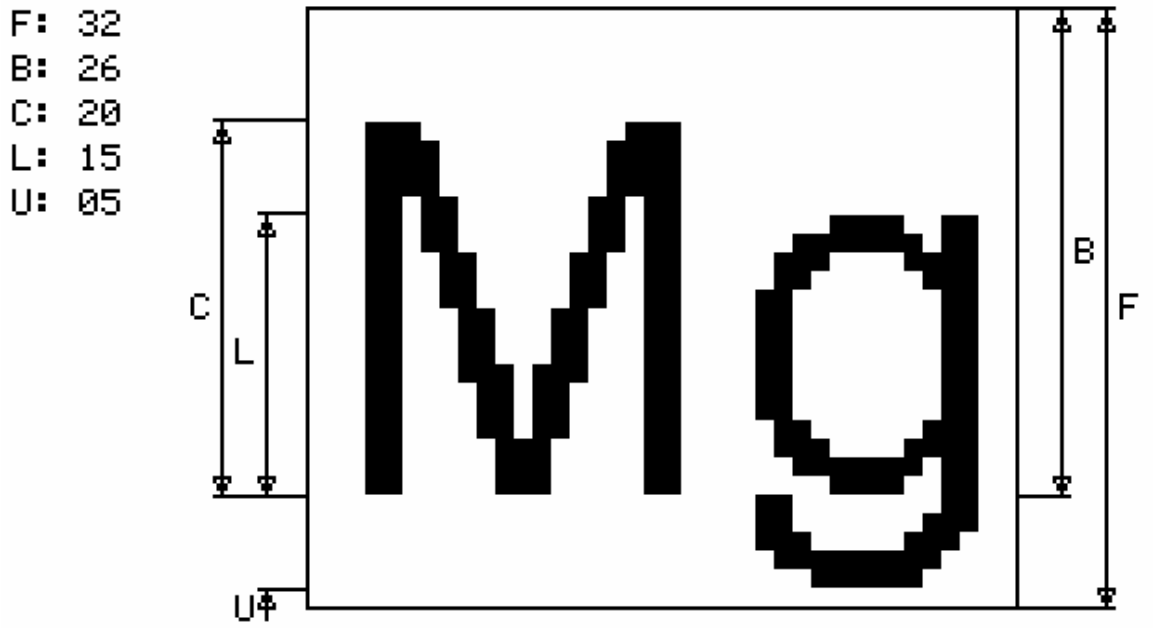
Font 24B – ISO 8859-1 (Latin1 or Western European)



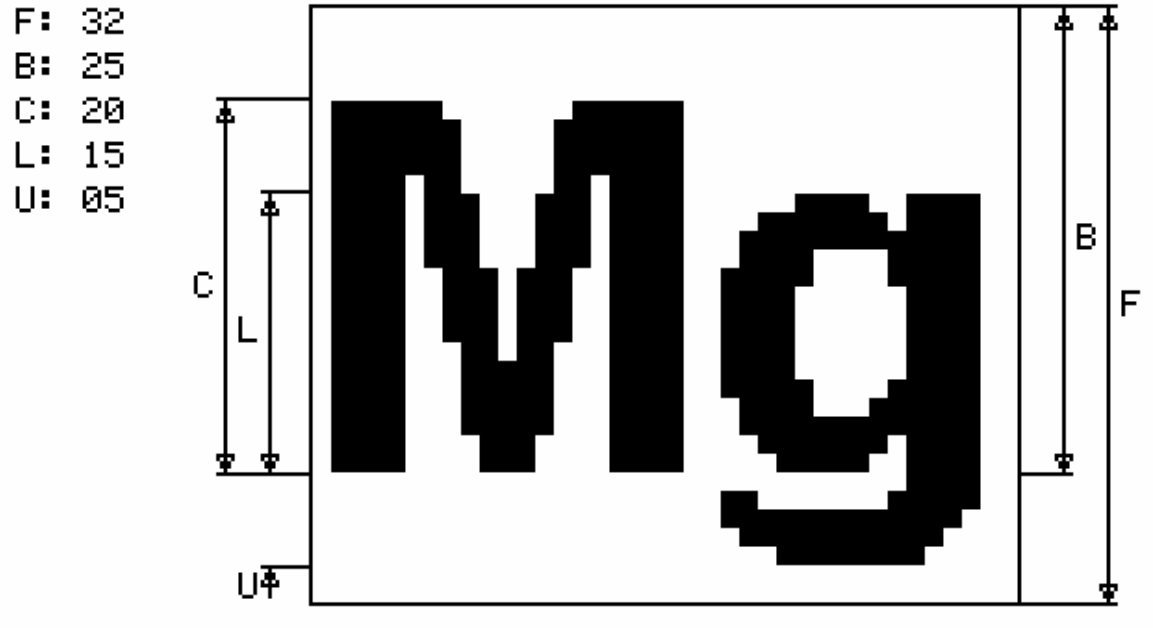
Font 24BC – ISO 8859-1 (Latin1 or Western European)



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



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



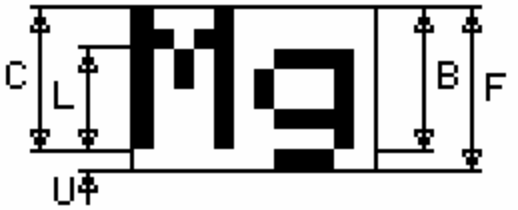
6.2. *Monospaced Fonts*

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

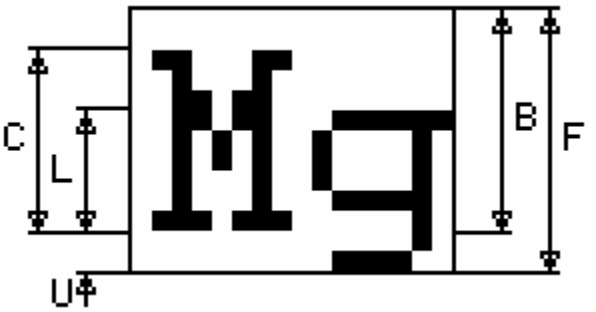
F: 10
B: 09
C: 09
L: 07
U: 01

Font 8x12 – ASCII Only

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

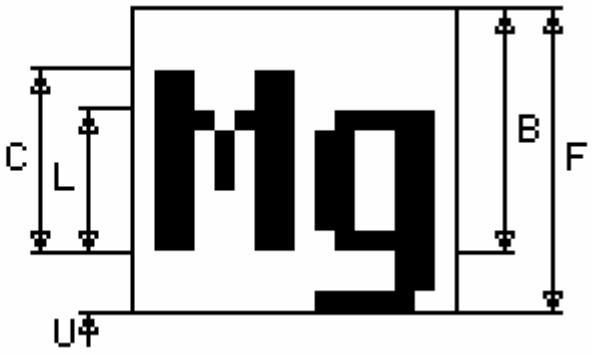
Font 8x13 – ASCII Only

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



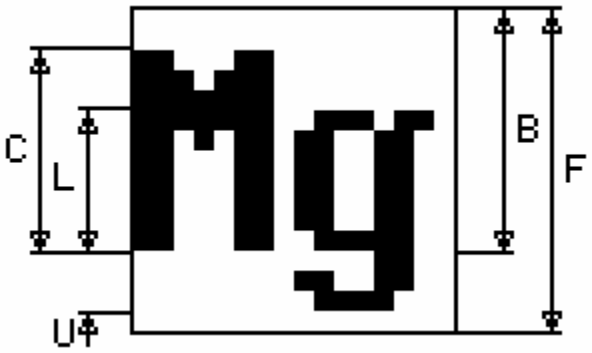
Font 8x15B – ASCII Only

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



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

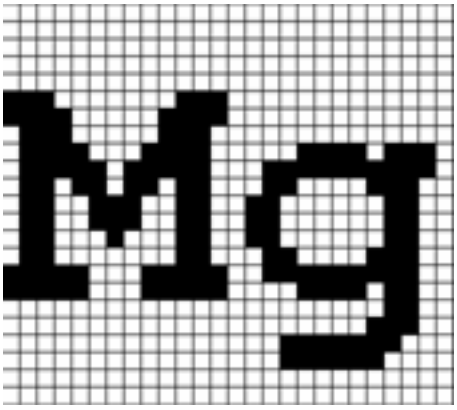
F: 16
B: 12
C: 10
L: 07
U: 03



Font 8x16L

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

Font 14x24 – ISO 8859-1



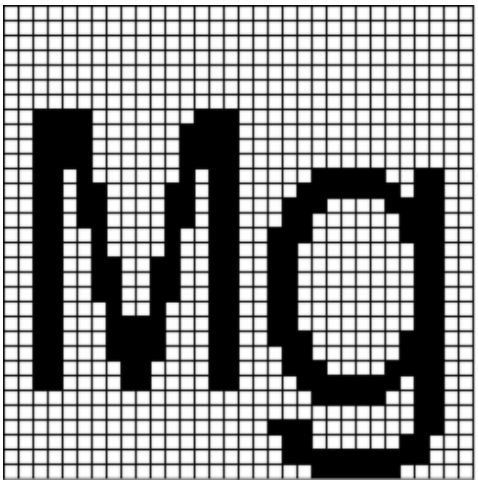
Font 16x32 – ISO 8859-1

This is the font 8x16 doubled in both directions:

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

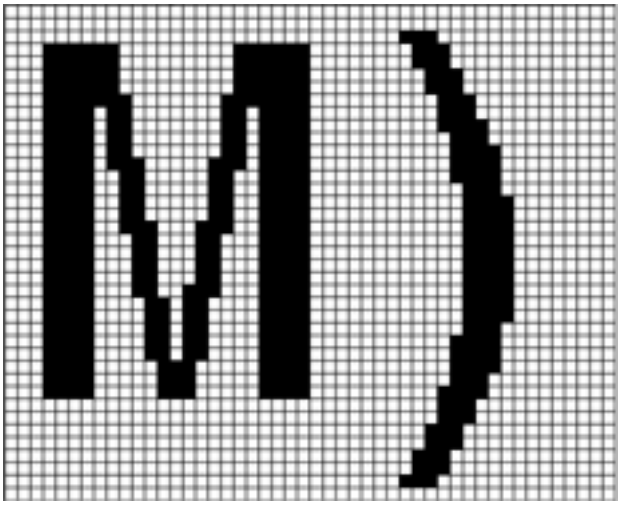
Font 16x32i – ISO 8859-1

This is an improved version of the 8x16 above.



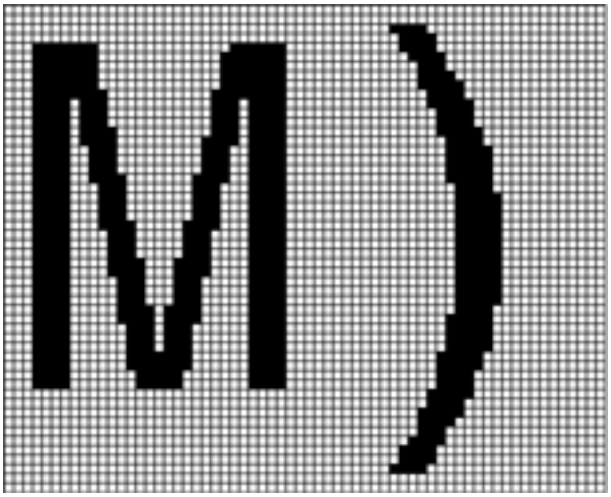
Font 24x48 – Numbers, Capital letters, Symbols

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



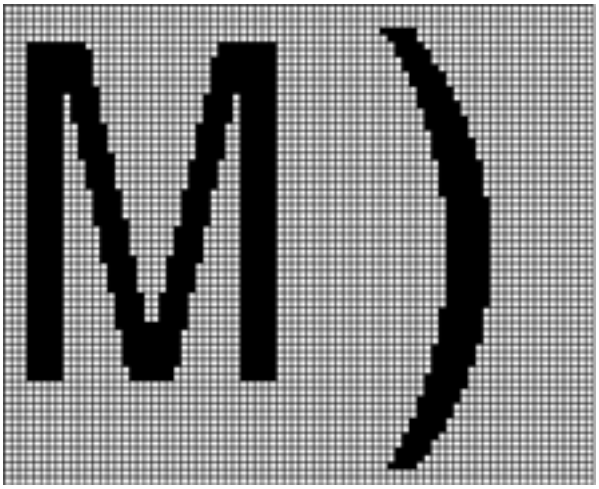
Font 32x64 – Numbers, Capital letters, Symbols

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



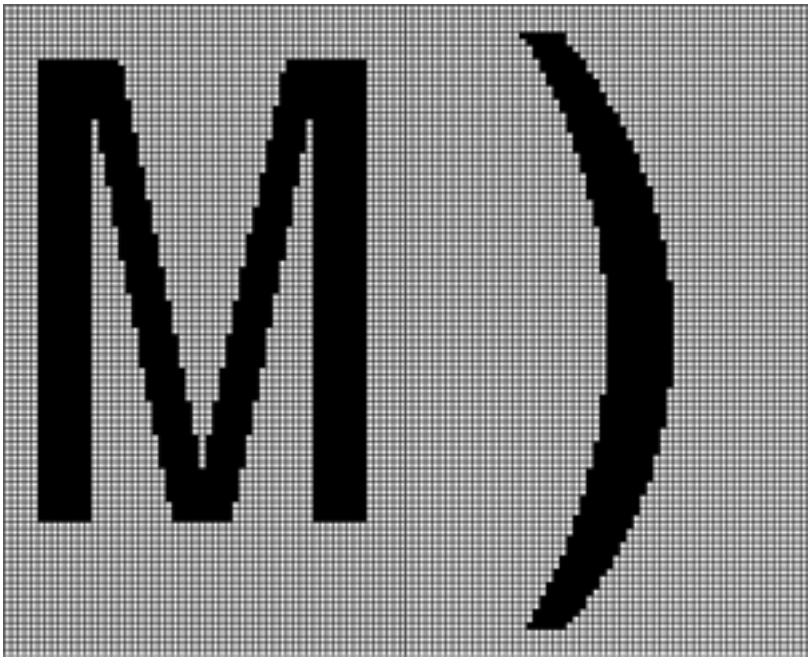
Font 40x80 – Numbers, Capital letters, Symbols

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

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

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

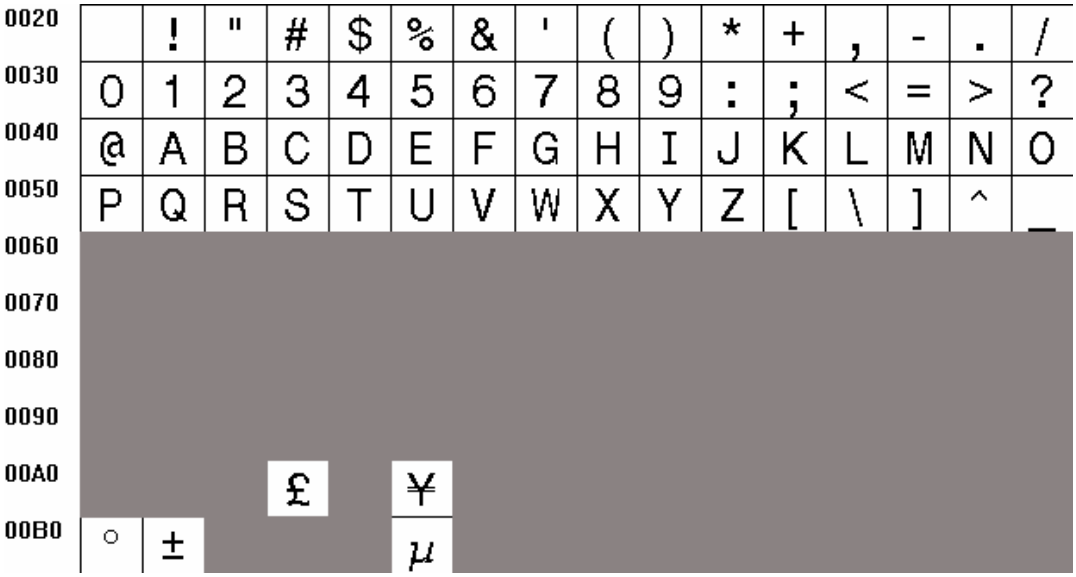
ú	250	uacute	u acute
û	251	ucirc	u circumflex
ü	252	uuml	u umlaut
ý	253	yacute	y acute
þ	254	thorn	thorn
ÿ	255	yuml	y umlaut

***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:



7. DEMO KIT TUTORIAL

7.1. *Connection and control via PC*

IMPORTANT: Before being able to send commands to the SLCD/6/43 you MUST remove the Demo jumper JP1 on the PowerCom 4 (triangle) board.

In order get acquainted with the SLCD commands, bitmap storage, and macro features, it is recommended that the kit be attached to a PC first. 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; however the BMPload program is Windows-only.

The two DB9 serial ports (Main and Aux) on the PowerCom4 board are 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 a PC serial port. Alternatively if you have a USB-serial adapter you can plug the adapter directly onto the PowerCom4 board. The Main port should be used for initial communications with the host PC.

This tutorial assumes only a basic PC installation is available and therefore uses Hyperterminal to communicate. Hyperterminal has significant limitations; the program REALTERM has been found to be useful in cases where Hyperterminal in unavailable or does not work. See http://realterm.sourceforge.net/

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. [Note the half-duplex description is confusing; the SLCD is full duplex but does not echo characters, so the half-duplex setting is needed.]

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 a new '>' 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. Alternatively, you can use two serial connections with BMPload on the second serial port connected to the "AUX" DB9.

7.2. *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. If they are not, use the BMPload program and the files on the CD in the "BMPs and Macros" folder.

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 both pressed and released:

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

7.3. *Macros*

The SLCD comes with pre-loaded macros to demonstrate this capability. Refer to the file "Macros.txt" 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.

7.4. *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 Elements, 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. See our new ***Download Center*** at http://www.reachtech.com. Sign up for an account and enjoy the wealth of technical information we have available for download.

8. WORKING WITH BITMAPS

8.1. *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 of the interface styles implemented in Microsoft Windows applications 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.

8.2. *8 bit Color Mode - SLCD controller only*

The SLCD (as opposed to the SLCD6 or SLCD43) only supports 8 bit color mode. 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 8 bit color bitmap, they are displayed more quickly if the bitmap's palette is the same as the SLCD's. To do this, 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.

The SLCD supports a custom palette as well. In this case, ensure that all bitmaps have the same palette, and use the "Custom Palette" option in the BMPload program.

8.3. *High Color Mode - SLCD6, SLCD43*

The SLCD6 and SLCD43 support high color mode as standard. These can both accept bitmaps with 1, 4, 8, and 24 bits per pixel. The BMPload program converts 24 bit BMPs into the RGB565 physical format used by the controller. In this format, 5 bits of color are used for RED and BLUE and 6 bits for GREEN.

The basic algorithm for converting between Truecolor (24-bit BMP file) to Highcolor (16-bit LCD screen) is below:

1. Divide [Blue and Red value] by 8 (ignore remainder)
2. Divide [Green value] by 4 (ignore remainder)

3. Add [Red value times 2048] plus [Green value times 32] plus [Blue value] (Highcolor value)

Users can read the stored Highcolor value by using the PIXEL READ command.

Note: The SLCD6 can support 8 bit color if needed for backward compatibility for SLCD developers..

9. RS485 MULTIPOINT COMMUNICATIONS

9.1. *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 254 SLCD controllers on a shared line; the actual limit may be less than this due to physical bus loading limitations.

9.2. *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 on the main port unless they are preceded by the RS485 address header. The main port autoswitch can be used to communicate with the controller using non-polled operation.

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.

`<AD2>` 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.

9.3. *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>
```

9.4. *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>
```

9.5. *RS485 half duplex vs. full duplex*

In half-duplex mode, transmit and receive are shared. The SLCD is naturally full duplex, so to use it in half duplex mode, it must ignore anything on the receive line while transmitting. This is effected by the SET HALF DUPLEX command as follows.

SET HALF DUPLEX

Description: This command tells the SLCD to ignore anything received while it is transmitting. This command sets this mode in non-volatile memory, and only needs to be executed once. This command should only be used in polled mode.

Command: `*com1Half`

10. IN-SYSTEM BITMAP AND FONT DOWNLOAD

10.1. *Introduction*

Starting with version 2.5 of the SLCD firmware, binary transfer of flash resident fonts, bitmaps and macros is supported. Binary transfers of screen data (Bitmaps) is also supported, either to an off-screen area, or directly to the display screen.

10.2. *Download flash image (bitmaps, macros, fonts)*

The SLCD flash memory can be updated in-system by using the binary download functionality. The image file is saved using the BMPload "Save to File" feature. To do this, use the command:

`bdld 4 0 <size> <timeout>`

where `<size>` is the number of bytes in the image file, and `<timeout>` is the timeout in milliseconds. The timeout is needed because once in binary transfer mode, the SLCD will not respond to normal commands.

Once the bdld command has been issued, the SLCD responds with a standard 2 character prompt '>',0x0d, and the transfer can begin. If successful when the transfer is complete another standard prompt will be issued. A timeout will generate a 2 character error prompt '!',0x0d/

The SLCD Flash memory must be erased before using command. The command is “xmc 0xFEEB”. This command is only needed with Binary Download command.

An optional user application algorithm feature is the use of checksums. An application can compare the stored checksum with an expected checksum (via BMPload application) using the CRC External Flash command. This will ensure the integrity of the downloaded data.

10.3. *Download and display image using off-screen memory*

EXAMPLE CODE:

```
void CBMPloadDlg::OnButtonDisplayImage()

{
        char cmdbuf[80] = {0};
        //wwr <x> <y> <x> <y> <index> <addr>
        //first x, y is upper left. last is bottom right. use index 0-3 for storage.
        //NOTE: because only the pixel data is stored,
        //any custom palette associated with the BMP file will be lost

        //display at upper left corner
        sprintf( cmdbuf, "wrr %d %d %d %d %d", 0, 0, storedBMPWidth, storedBMPHeight, 0 );
        //if drop the 6th parameter (address), will default to 0
```

```
        m_SerialPort.WriteLine( cmdbuf );

        //display to the left of first image
        sprintf( cmdbuf, "wrr %d %d %d %d %d",
                storedBMPWidth, 0, storedBMPWidth, storedBMPHeight, 0 );

        m_SerialPort.WriteLine( cmdbuf );

        //display to the left of second image with 1 pixel gap and start writing at
        //beginning address + half the width.
        //since we have one image in storage the beginning address is 0.
        //since we have an address offset, the last pixels will display data outside the
        //range of the current image
        //because the binary download uses images that are 8 bits per pixel, if we had a
        //second image it's beginning storage address would be image 1 width * image 1
        //height

        sprintf( cmdbuf, "wrr %d %d %d %d %d %d",
                storedBMPWidth*2 + 1, 0,
                storedBMPWidth, storedBMPHeight, 0 , 0 + storedBMPWidth/2 );

        m_SerialPort.WriteLine( cmdbuf );

        //display at 100, 100
        sprintf( cmdbuf, "wrr %d %d %d %d %d", 100, 100,
                storedBMPWidth, storedBMPHeight, 0 );

        m_SerialPort.WriteLine( cmdbuf );

        //display another below the first image and simulate a marquee.
        //since we have an address offset, the last pixels will display data outside the
        //range of the current image.

        for(int i = 0; i <= storedBMPWidth; i++){
                sprintf( cmdbuf, "wrr %d %d %d %d %d %d", 0, storedBMPHeight,
                        storedBMPWidth, storedBMPHeight, 0 , i );

                m_SerialPort.WriteLine( cmdbuf );
                Sleep(10);
        }
        for(i = storedBMPWidth; i >= 0; i--){
                sprintf( cmdbuf, "wrr %d %d %d %d %d %d", 0, storedBMPHeight,
                        storedBMPWidth, storedBMPHeight, 0 , i );
                m_SerialPort.WriteLine( cmdbuf );
                Sleep(10);
        }
}

int CSerial::ExtMemProgramBin(BYTE * buf, int bytes, CEdit * status, CDialog * dlg)
{
        CString str;
        int result=0;
        unsigned long sent;
        char cmdbuf[80];
        int secs;
        int bytes_remaining;
        int percent;
        int last_percent = 0;
        BYTE * buf_ptr;
        MSG msg;

        buf_ptr = buf;
        bytes_remaining = bytes;

        if (!LocateDevice())
        {
                if (dlg)
                        dlg->MessageBox("Device not responding",
                        "Programming",MB_ICONERROR);
                return 0;
        }
```

```
	//Write to windows save/restore memory (index 0-3). use index 0 for demo
	//bdld <index><address offset><length in bytes><timout(ms)>

	sprintf( cmdbuf, "bdld 0 0 %d 2000", bytes );
	WriteLine( cmdbuf );

	// wait for an ACK from panel.
	for (secs=0;secs<5*SERIAL_SEC_MULT && !*m_Buffer;secs++)
	{
		m_Buffer[0]='\0';
		ReadLine(m_Buffer,BUFFER_SIZE);
	}

	while(bytes_remaining)
	{
		result=WriteFile(m_hComm, buf_ptr, MIN(MAXBYTES,
			bytes_remaining), &sent, NULL);
		buf_ptr += MAXBYTES;
		bytes_remaining -= MIN(MAXBYTES, bytes_remaining);
		percent= 100 - (bytes_remaining * 100) / bytes;
		if( last_percent != percent || ( bytes_remaining == 0 ) )
		{
			while (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
			{
			// the only way out of the loop

			  if(msg.message == WM_QUIT) break;
			  TranslateMessage(&msg);
			  DispatchMessage(&msg);
			}

			// check input buffer for fail from panel programming

			ReadChars(m_Buffer,BUFFER_SIZE);
			if( m_Buffer[0] == '!' )
			{
				str.Format("Programming %u Bytes, ***FAILED***", bytes,
percent);
				status->SetSel(0,-1);
				status->ReplaceSel(str);
				return ( 0 );		// return failure
			}

			str.Format("Programming %u Bytes, %d %% Completed...", bytes,
percent);
			status->SetSel(0,-1);
			status->ReplaceSel(str);
			last_percent = percent;
		}
	}

	Flush();

	return result;
}
```

11. USING CRC'D COMMANDS

11.1. *Overview*

As of version 2.6.29, the SLCD can accept a command with a CRC prefix and use it to verify the command was not corrupted in transmission from the host. Once verified, the command is processed in the normal manner, and the SLCD responds as expected. If, however, the CRC check fails, the SLCD ignores the command and returns a invalid CRC response ('#'<return>).

11.2. *Command Protocol*

The format for a CRC'd command is:

`~<CRC><SLCD Command><return>`

A '~' (tilde) character at the start of the command string signals the SLCD that an embedded CRC (4 ASCII-Hex chars, [0-9,a-f,A-F]) will follow the '~' and then the actual SLCD command will begin. The CRC is calculated for the SLCD command and its <return>, which means a NULL Command (just a <return>) will still have a CRC to validate the <return>.

For example: to send the "s 0 1" command with a CRC, calculate the CRC for the 'C' string "s 0 1\r", which is 0x9ACB. Send:

`~9ACBs 0 1<return>`

and the SLCD will validate the command, execute it, and respond with the '>'<return> prompt, indicating success. If the CRC value does not match the string's computed CRC, the '#'<return> prompt is given. If the CRC is correct, but the command has a syntax error, the standard error prompt '!'<return> is given.

11.3. *Example CRC generation code*

Included below is 'C' code for a program that accepts a standard SLCD command as input and generates the CRC'd version of the command as output. It includes a CRC generator function that produces CRC's compatible with the SLCD. The CRC polynomical is CRC-CCITT.

```
#include <stdio.h>
#include <string.h>

//================================================================
// CRC calculation routine
//
// CRC argument allows you to accumulate
// the CRC value over multiple buffers
//================================================================

const unsigned short crctableA[ 16 ] =
{
    0x0000, 0x1081, 0x2102, 0x3183, 0x4204, 0x5285, 0x6306, 0x7387,
    0x8408, 0x9489, 0xA50A, 0xB58B, 0xC60C, 0xD68D, 0xE70E, 0xF78F
};
```

```
const unsigned short crctableB[ 16 ] =
{
    0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
    0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7
};

unsigned short crc16(unsigned char *address, unsigned int size, unsigned short crc)
{
    for (; (size > 0); size--)
    {
        /* byte loop */
        unsigned char data = *address++; /* fetch the next data byte */

        data ^= crc; /* EOR data with current CRC value */
        crc = ((crctableA[(data & 0xF0) >> 4] ^ crctableB[data & 0x0F]) ^ (crc >> 8));
    }
    return(crc);
}

//====================================================================
// main()
//====================================================================

static char cmdStr[ 129 ];

// syntax: CmdCrc "SLCD Command Line<CR>"
// output: "~HHHHSLCD Command Line<CR>"
int main(int argc, char* argv[])
{
    unsigned short crc = 0xFFFF;

    // must be 1 and only 1 arg:
    if( argc == 2 )
    {
        // copy slcd command to our buffer:
       strcpy( cmdStr, argv[1] );
       // append a <CR>:
       strcat( cmdStr, "\r" );
       // calc the CRC:
       crc = crc16( (unsigned char *)cmdStr, strlen(cmdStr), crc );
       // show results:
       printf( "   Input:  [%s\\r]\r\n", argv[1] );
       printf( "   Output: [~%04X%s\\r]\r\n", crc, argv[1] );
    }
    else
    {
       printf( "   ERROR: syntax is 'CmdCrc \"slcd cmd\"' (quotes req'd)\r\n" );
       return( -1 );
    }
    return 0;
}
```

Example usages:

```
C:\CRC>cmdcrc "s 0 1"
   Input:  [s 0 1\r]
   Output: [~9ACBs 0 1\r]


C:\CRC>cmdcrc "t \"Hello\""
   Input:  [t "Hello"\r]
   Output: [~9F42t "Hello"\r]
```

12. SOFTWARE MANUAL CHANGE HISTORY

5/11/2008	Initial version of the Software Manual for SLCD, SLCD6, and SLCD43 controller boards.
6/18/2008	- Added note to xio command with high color firmware - Changed BMPload to 1.7.7 with added optional sort - BMPload save list is now filenames only, not directory/filename so the list files can be relative. - BMPload now supports option to sort bitmaps as loaded. - Renamed "16 bit color" mode to "high color mode" to reduce confusion; high color mode uses 24 bit BMP files and reduces them to 16 bit for storage and display.
6/30/08	Additional details of new VERSION command update. Added screen shot for 1.8.0. This is the official external release.
7/9/08	Additional note on “xio” command.
7/17/08	Button Define command, button numbers 118-127 support “long strings” of a length of 50 characters, rather than the default of 20.
7/18/08	Added command “Button Define Center Text”.
7/23/2008	Added Draw Filled Ellipse.
7/24/2008	Added comments on Font name alias size. Made notes more consistent in command description section.
7/28/2008	Added Chart Bitmap Define.
7/30/2008	Added Command Debug.
9/4/2008	Added additional comments for the Binary Download command and “Download flash image (bitmaps, macros, fonts)” section.
11/20/2008	Minor clarifications to WINDOW RESTORE RECTANGLE command.
02/09/2009	Added macro Labels explanation and TOUCH MACRO ASSIGN label information. Added commands DEFINE DISPLAYABLE CURSOR, SET DISPLAYABLE CURSOR POSITION, TOUCH DISPLAYABLE CURSOR
02/10/2009	Added additional Internal Arguments: Integer Internal Variable, String Internal Variable, Point Coordinate Internal Variable. Added commands SET VARIABLE, GET VARIABLE.
2/18/2009	Missing command syntax for DEFINE DISPLAYABLE CURSOR in introductory tables. Extra DEFINE DISPLAYABLE CURSOR command removed in SOFTWARE COMMAND REFERENCE section.
4/01/2009	Added Section 11 CRC Command feature and example code; fixed DISPLAY CLIPPED BITMAP IMAGE command description and examples.
4/03/2009	Added STRIP type to CHART DEFINE command.
4/14/2009	Cleaned things up and added new features to support the 2.7.0 general release.
4/21/2009	Additional issues fixed from our database: mpush and mpop arguments corrected and better examples; Removed colon from host notification for BUTTON

	DEFINE – LATCHING STATE; Macro parameter escape character defined more clearly; Removed reference to “true Host” capabilities (we no longer support); Added note to Define Hotspot commands about clearing pre-existing hotspots.
4/24/2009	Added BMPload screen shot to 1.9.0 and Orientation feature.