SLCD+/6/43 Command Reference Manual

SLCD+, SLCD6, SLCD43 Firmware Version 2.10.0
BMPload Version 1.12.0

07/27/2022

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

Reach Technology
www.reachtech.com

Sales
503-675-6464 ext. 2
sales@reachtech.com

Technical Support
503-675-6464 ext. 1
techsupport@reachtech.com

Table of Contents

0. HARDWARE LIMITED WARRANTY AND SOFTWARE LICENSE AGREEMENT

0.1 Hardware Limited Warranty

REACH TECHNOLOGY, a Novanta Company, 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 supersedes 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, 4600 Campus Place, Mukilteo, WA 98275 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, a Novanta Company (“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 AT: REACH TECHNOLOGY, 545 First Street, Lake Oswego, OR 97034.

Reach Technology ("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 ultra-hazardous 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 Supported Hardware (model, color mode, and pixel resolution)

This manual pertains to the SLCD+, SLCD6, and SLCD43 controllers. These will be referred to generically as SLCDx controllers.

All SLCDx controllers support Highcolor mode as a standard. The SLCD+ and SLCD6 are also available with 8-bit color mode firmware to support products using the legacy SLCD controller (which is no longer available).

The SLCD+ and SLCD6 have QVGA resolution (320x240 pixels), whereas the SLCD43 has WVGA resolution (480x272 pixels).

1.2 General

An SLCDx controller and attached LCD touch panel make up a module, generically referred to as an SLCDx. An SLCDx 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 SLCDx and receives button press responses from the SLCDx. In this manual, the term "host" is used to describe the device connected to the SLCDx.

The SLCDx 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 SLCDx 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 SLCDx. Macros are a sequence of SLCDx commands and are described in Section 4.

The SLCDx is connected to the host processor via a serial port. The number of serial ports is SLCDx 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 (only 1 is in control at any given time). The PC can also be used for interactive command execution / testing.

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

1.3 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, all SLCDx support Highcolor mode as standard, and 8-bit color mode is available on the SLCD+ and SLCD6 to support products migrating off the older, legacy SLCD. For most new designs, Highcolor bitmaps should be used. *Note that Highcolor 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 WORKING WITH BITMAPS.

Bitmaps are numbered from 1 to n, depending on how many bitmaps are loaded (with BMPload). The first bitmap loaded will be bitmap #1, the second bitmap loaded will be #2, etc.

1.4 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 SLCD6 and SLCD43 have 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. ***Warning: the Belkin USB-serial adapter has software compatibility issues and is not recommended.***

Case Sensitivity

All commands are case-sensitive as shown in this manual. The meaning of some commands changes depending on the use of upper- or lower-case.

[1] Note: Flow control is supported on SLCDx receive only; that is, the SLCDx 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.

1.5 SLCDx Input Buffer Processing

Input Buffer

The SLCDx 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 SLCDx 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 SLCDx will issue a prompt for each command after it executes. These prompts may be issued while the host is sending a command to the SLCDx (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 SLCDx 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 SLCDx.

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 command (*abt<return>) and look for a received success prompt

If the SLCDx 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 SLCDx 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 SLCDx 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 SLCDx 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 SLCDx. 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 SLCDx. The limit event can occur when the buffer is full of unexecuted commands.

In both of the previous cases, when the SLCDx 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 SLCDx 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.

If the display returns an error mid-command, then it probably got a bit error. If it returns an error after receiving the <return>, then it is probably getting either a CRC error or a syntax error. Either of these could be from getting a message with missing bytes or incorrect bytes.

1.6 Touch Interface

The SLCDx 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 SLCDx can either notify the host directly or execute a "macro", or both. A macro is a predefined sequence of SLCDx commands.

Hotspot

The term “hotspot” refers to an area of the display that is touch-sensitive. There are two types of “hotspots”: visible and invisible. A visible hotspot is the standard type and when touched, the display area of the hotspot is color inverted (technically XOR’d with the foreground color) to provide a visual indication that a hotspot has been activated. An invisible hotspot does not provide any visual indication when touched. See SET TOUCH CHARACTERISTICS for information on hotspot touch characteristics (notify on press, release, typematic, etc).

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. See SET TOUCH CHARACTERISTICS for information on hotspot touch characteristics (notify on press, release, typematic, etc).

Host Notification

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

1.7 Host Input Processing

When integrated into a host environment, the SLCDx sends prompts, touch activity notifications, and user-defined text to the host it is connected to. In general, all SLCDx 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.

1.8 Control Port Autoswitch

For added flexibility, the main control port of the SLCDx can be switched on the fly to any of the other serial ports. This is called “control port autoswitch”. This is done 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

NOTE: ♦ Indicates that the command stores the setting in EEPROM (non-volatile); see the warning concerning these commands.

Function	Command Name	Command
Animation	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`
	Wait for Refresh	`wrf <x> <y>`
	Wait Vertical Retrace	`wvr <line> [<line2>]`
Bitmaps / Splash Screen	Display Bitmap Image	`xi <index> x y`
	Display Bitmap Image Centered	`xim <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>`
	List Bitmaps Detail	`lsbmp [index]`
	Splash Screen♦	`*SPL <number>`
Buttons / Touch	Allow Pre-existing Touch	`*apt`
	Allow Pre-existing Touch, No Beep	`*aptnb`
	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`
	Clear All Hotspot	`xc all`
	Clear Hotspot	`xc <n>`
	Define Displayable Cursor	`curs <bitmap index> <transparent color index>`
	Define Hotspot	`x <n> x0 y0 x1 y1`

	Define Relative X-Y Hotspot	xxy <n> x0 y0 x1 y1
	Define Relative X-Y Hotspot (Silent – No Beep)	xxynb <n> x0 y0 x1 y1
	Define Special Hotspot (Invisible)	xs <n> x0 y0 x1 y1
	Define Special Hotspot (Invisible, No Beep)	xsnb <n> x0 y0 x1 y1
	Define Special Typematic Touch	xst <n> x0 y0 x1 y1
	Define Touch Action♦	*touchMode [S|L|W]
	Define Touch Parameters♦	*touchParm [<samples>]
	Define Typematic Touch Area	xt <n> x0 y0 x1 y1
	Disable Touch	xd <n | "all"> [<n last>]
	Enable Touch	xe[s] <n | "all"> [<n last>]
	Query (Latching) State Button	ssb <n>
	Reset Touch Calibration♦	*RT
	Set Displayable Cursor Position	curp x y
	Set State Button Delimiter	*sdelim
	Set Touch Characteristics	xset <n> [+|-][p|r|t|T|x]
	Set Touch Debounce♦	*debounce <delay in ms>
	Set (Latching) State Button	ssb <n> state
	Set Typematic Parameters♦	typematic[s] <delay> <repeat>
	Simulate Touch	*simtouch x y
	Touch Calibrate♦	Tc
	Touch Displayable Cursor	curt
	Touchscreen On/Off	touch [on/off]
Charts	Binary Chart Values	bcv n number_of_pen_values_to_follow
	Chart Clear Display	cc n
	Chart Define	cd n x0 y0 x1 y1 t dw bv tv bc <pens>
	Chart Define with Bitmap	cdb n x y dw bv tv bitmap <pens>
	Chart Redefine Backgrd Bitmap	cdbb n bitmap
	Chart Redefine Backgrd Color	cdbc n color
	Chart Redefine Draw Mode	cddm n <on|off>
	Chart Redefine Pen	cdp n pen width color
	Chart Redefine Retrace Mode	cdrm n <on|off>
	Chart Reset Pens	cr n
	Chart Values	cv n pen0_value [pen1_value ..]
Clear / Unclear	Clear All Hotspot	xc all
	Clear Hotspot	xc <n>
	Clear Screen	z
	Clear Screen Special	zs <bitmap index>
	Screen Blank (Basic)	sb color
	Screen Blank (Complete)	SB <color_detail>

	Screen Unblank (Basic)	su
	Screen Unblank (Complete)	SU
Com Port	Communication Watchdog Timer	*comwdt <macro> <short> <long>
	Control Port Autoswitch♦	(see description)
	Output String (Aux)	aout "<text string>"
	Output String (Main)	out "<text string>"
	Query Control Port	*com?
	Read from Aux Port N (N = 0, 1, 2, or 3 for SLCD6/43) (N = 0, or 1 for SLCD+)	ainN
	Set Aux Escape♦	*auxEsc <hex value of ASCII character>
	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 | 4800 | 2400 | 1200 | 300]
	Set Control Port♦	*com[0|1|2|3]main[s]
	Set Previous Control Port	*prevCons
	Write to Aux Port N	aoutN "<text string>"
Debug / Maint.	Color Test	*TESTC
	Debug Command	*cmddebug <0|1>
	Debug Macro	*macdebug <0|1>
	Debug Touch	*debug <0|1>
	Reset Board / Software	*RESET
	Reset Board to Mfg. State♦	*MFGRESET
	Speed Test	*speedtest
Drawing Environ-ment	Check Color Mode	*IS16
	Get Pen Width	p
	Set Color (Custom Palette)	s <fore> <back>
	Set Color (Basic)	s <fore> <back>
	Set Color (Detailed)	S <foreground_det> <background_det>
	Set Draw Mode	d [n|x]
	Set Origin	o <x> <y>
	Set Pen Width	p <pixels>
	Restore Drawing Environment	sr
	Save Drawing Environment	ss
	Window Restore	wr x y [index]
	Window Restore Rectangle	wrr x y <width> <height> <index> [<offset>]
	Window Save	ws x0 y0 x1 y1 [index]
Download	Binary Download	bdld <index> <offset> <size> <timeout>

	Binary BMP Download	bbmp <index> x0 y0 x1 y1 [timeout]
	External Memory Available	xma
	External Memory Chip Erase	xmc FEEB
	External Memory Block Erase	xme <addr> FEEB
Macros	Debug Macro	*macdebug <0|1>
	Demo	Demo
	Get Variable	get <internal variable name>
	Get Variable (HEX)	getx <internal variable name>
	List Macros Detail	lsmac [index]
	Macro Abort	*abt
	Macro Execute	m <n> [macro parameters . .]
	Macro Notify	*macnote <0|1|2|3>
	Memory Pop	mpop [index] <number>
	Memory Push	mpush [index] "<string>" [max]
	Power-On Macro♦	*PONMAC <index | name> [<option>]
	Set Demo Macro♦	*DEMOMAC <index | name>
	Set Variable	set <internal variable name> <value>
	Set Variable (HEX)	setx <internal var. name> <HEX value>
	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>
	Wait	w <number of milliseconds>
Read / Write / Verify	CRC External Flash	*CEXT [<from> <to>]
	CRC Processor Code	*CSUM
	CRC Screen	*CRC
	EEPROM Read	*eer <hex location>
	EEPROM Write♦	*eew <hex location> <hex value>
	Get Panel Type	*panel
	I2C Read	i2cr <addr> <reg> <length>
	I2C Speed	i2c speed <speed>
	I2C Write	i2cw <addr> <reg> <data>
	List Downloaded Records	ls
	Read Frame Buffer Line	*FB <line>
	Read LCD Controller	XR <hex register>
	Read Temperature	temp
	Set I/O Pin	iopin
	Set LED	led [0|1]
	Version	vers

	Write LCD Controller	XW <hex register> <hex value>
Screen Control	Display On/Off	v <on|off>
	External Backlight Bright Ctrl♦	xbb [+|-]<level>
	External Backlight Bright Ctrl	xbbs[f] [+|-] <level>
	External Backlight On/Off	xbl <on|off>
	Set Panel Orientation♦	*orient [0|1]
Shapes / Pixels	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 Rotated Filled Polygon	pfr <angle> x0 y0 [x/y x/y...]
	Draw Rotated Polygon	pgr <angle> x0 y0 [x/y x/y...]
	Draw Rotated Polyline	plr x0 y0 [x/y x/y...]
	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]
	Redraw Rotated Polygon	ppgr <angle> x0 y0
	Draw Triangle	tr x0 y0 x1 y1 x2 y2 [color]
	Pixel Read	pr
	Pixel Write	pw x y [color]
Sliders Meters & Levelbars	Binary Notification Mode	*binr <0|1>
	Levelbar Define	ld n x0 y0 x1 y1 or inv bv bc <levels>
	Levelbar Value	lv n val
	Meter Define	md <id> <bitmap> <x> <y> <type> <minVal> <maxVal> <initial_value> <minAngle> <maxAngle> <x0, y0 … [x10, y10]>
	Meter Define Band	mdb <ix> <bm> <x> <y> <type> <minVal> <maxVal> <init_val> <minAngle> <maxAngle> <bandMin> <bandMax> <bandRadius> <bandPen > <bandColor> <bandBG> <pivotX> <pivotY> <x1 y1 . . .[x10, y10]>
	Meter Value	mv <id> <value>
	Meter Value Band	mvb <ix> <val> <bandMin> <bandMax> <bandRadius> <bandPen> <bandColor> <indicatorColor>

	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
Sound	Alarm	al <alarm> <count>
	Beep Frequency♦	bf[s] [<hertz>]
	Beep Once	beep <count>
	Beep Repeat	rb <on> <off> [alarm]
	Beep Touch	bb <number>
	Beep Volume♦	bv[s] [+|-]<level>
	Beep Wait	beepw <count>
Text	Append to Scrolling Textbox	sta <index> "text"
	Clear Scrolling Textbox	stc <index>
	Define Scrolling Textbox	std <index> <type> <x0> <y0> <x1> <y1>
	Prepend to Scrolling Textbox	stp <index> "text"
	Query Scrolling Textbox	std <index>
	Get Text Display Width In Pixels	t? "text string"
	Set Cursor	sc x y
	Set Font	f <fontName>
	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 Clear	tfc
	Text Flashing Delete	tfx <index>
	Text Flashing Disable	tfd <index> <state>
	Text Flashing Display	tf <index> [t] "text string" x y [R|T|X|TR]
	Text Flashing Enable	tfe <index>
	Text Flashing List	tf? <n>
	Text Flashing Synchronize	tfs
	UTF8 Enable / Disable	utf8 [on|off]

Commands Alphabetically by Command Syntax

NOTE: ♦ Indicates that the command stores the setting in EEPROM (non-volatile); see the warning concerning these commands

Command	Command Name
*abt	Macro Abort
*apt	Allow Pre-existing Touch
*aptnb	Allow Pre-existing Touch, No Beep
*auxEsc <hex value of ASCII character>	Set Aux Escape♦
*binr <0|1>	Binary Notification Mode
*CEXT [<from> <to>]	CRC External Flash
*cmddebug <0|1>	Debug Command
*com[0|1|2|3]main[s]	Set Control Port♦
*com?	Query Control Port
*comwdt <macro> <short> <long>	Communication Watchdog Timer
*CRC	CRC Screen
*CSUM	CRC Processor Code
*debounce <delay>	Set Touch Debounce♦
*debug <0|1>	Debug Touch
*DEMOMAC <index>	Set Demo Macro♦
*eer <hex location>	EEPROM Read
*eew <hex location> <hex value>	EEPROM Write♦
*FB <line>	Read Frame Buffer Line
*IS16	Check Color Mode
*macdebug <0|1>	Debug Macro
*macnote <0|1|2|3>	Macro Notify
*MFGRESET	Reset Board to Manufactured State♦
*orient [0|1]	Set Panel Orientation♦
*panel	Get Panel Type
*PONMAC <index> [<option>]	Power-On Macro♦
*prevCons	Set Previous Control Port
*RESET	Reset Board / Software
*RT	Reset Touch Calibration♦
*sdelim	Set State Button Delimiter
*simtouch x y	Simulate Touch
*speedtest	Speed Test
*SPL <index>	Splash Screen♦
*TESTC	Color Test
*touchMode[s] [S|L|W]	Define Touch Action♦
*touchParm [<samples>]	Define Touch Parameters♦
<return> <return> <return> 3 consecutive <return> characters to the Aux port	Control Port Autoswitch

a x0 y0 <radius> <start> <end>	Draw Arc Segment
ainN	Read From Aux Port N
al <alarm> <count>	Alarm
ani <n> <text string>	Animation Define
ani? <n>	Animation List
anic	Animation Clear
anid <n> [yield #]	Animation Disable
anie <n>	Animation Enable
anis	Animation Synch
anix <n>	Animation Delete
aout "<text string>"	Output String (Aux)
aoutN "<text string>"	Write to Aux Port N
baudN [230400|115200|57600|38400|19200|9600]	Set Baud Rate ComN
bb <number>	Beep Touch
bbmp <index> x0 y0 x1 y1 [timeout]	Binary BMP Download
bc <n>	Button Clear
bcv n number_of_pen_values_to_follow	Binary Chart Values
bd <n> x y type "text" dx dy bmp0 bmp1	Button Define – Momentary
bd <n> x y type "text0" "text1" dx0 dy0 dx1 dy1 bmp0 bmp1	Button Define – Latching State
bdc <n> x y type "text0" ["text1"] bmp0 bmp1	Button Define Center Text
bdld <index> <address> <size> <timout>	Binary Download
beep <count>	Beep Once
beepw <count>	Beep Wait
bf[s] [<hertz>]	Beep Frequency♦
bv[s] [+|-]<level>	Beep Volume♦
c x0 y0 r [f]	Draw Circle
cc n	Chart Clear Display
cd n x0 y0 x1 y1 t dw bv tv bc <pens>	Chart Define
cdb n x y dw bv tv bitmap <pens>	Chart Define with Bitmap
cdbb n bitmap	Chart Redefine Background Bitmap
cdbc n color	Chart Redefine Background Color
cddm n <on|off>	Chart Redefine Draw Mode
cdp <n> <pen> <width> <color>	Chart Redefine Pen
cdrm n <on|off>	Chart Redefine Retrace Mode
cr n	Chart Reset Pens
curp x y	Set Displayable Cursor Position
curs <bitmap index> <transparent color index>	Define Displayable Cursor
curt	Touch Displayable Cursor
cv n pen0_value [pen1_value ..]	Chart Values
d [n|x]	Set Draw Mode
Demo	Demo

dp x y	Draw Point
e x y <x radius> <y radius>	Draw Ellipse
ef x y <x radius> <y radius>	Draw Filled Ellipse
f <type>	Set Font
get <internal variable name>	Get Variable
getx <internal variable name>	Get Variable (HEX)
i <number> x y	Display OEM Bitmap Image
i2c speed <speed>	I2C Speed
i2cr <addr> <reg> <length>	I2C Read
i2cw <addr> <reg> <data>	I2C Write
iopin	Set I/O Pin
k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]	Scroll Screen Area
l x0 y0 x1 y1	Draw Line
ld n x0 y0 x1 y1 or inv bv bc <levels>	Levelbar Define
led [0|1]	Set LED
ls	List Downloaded Records
lsbmp [index]	List Bitmaps Detail
lsmac [index]	List Macros Detail
lv n val	Levelbar Value
m <n> [macro parameters . .]	Macro Execute
md <id> <bitmap> <x> <y> <type> <minVal> <maxVal> <initial_value> <minAngle> <maxAngle> <x0, y0 … [x10, y10]>	Meter Define
mdb <ix> <bm> <x> <y> <type> <minVal> <maxVal> <init_val> <minAngle> <maxAngle> <bandMin> <bandMax> <bandRadius> <bandPen > <bandColor> <bandBG> <pivotX> <pivotY> <x1 y1 . . .[x10, y10]>	Meter Define Band
mpop [index] <number>	Memory Pop
mpush [index] “<string>” [max]	Memory Push
mv <id> <value>	Meter Value
mvb <ix> <val> <bandMin> <bandMax> <bandRadius> <bandPen> <bandColor> <indicatorColor>	Meter Value Band
o <x> <y>	Set Origin
out "<text string>"	Output String (Main)
P	Get Pen Width
P <pixels>	Set Pen Width
pf x0 y0 [x/y x/y ...]	Draw Filled Polygon
pfr <angle> x0 y0 [x/y x/y...]	Draw Rotated Filled Polygon
pg x0 y0 [x/y x/y...]	Draw Outline Polygon
pgr <angle> x0 y0 [x/y x/y...]	Draw Rotated Polygon

`pl x0 y0 [x/y x/y...]`	Draw Polyline
`plr x0 y0 [x/y x/y...]`	Draw Rotated Polyline
`ppgr <angle> x0 y0`	Redraw Rotated Polygon
`pr`	Pixel Read
`pw x y [color]`	Pixel Write
`r x0 y0 x1 y1 [style] [color]`	Draw Rectangle
`rb <on> <off> [alarm]`	Beep Repeat
`S <fore_detail> <back_detail>`	Set Color (Detailed)
`s <fore> <back>`	Set Color (Basic)
`SB <color_detail>`	Screen Blank (Complete)
`sb color`	Screen Blank (Basic)
`sc x y`	Set Cursor
`set <internal variable name> <value>`	Set Variable
`setx <internal variable name> <HEX value>`	Set Variable (HEX)
`sl idx bg x y slider off ornt inv cont hi lo`	Slider Define
`sr`	Restore Drawing Environment
`ss`	Save Drawing Environment
`ssb <n>`	Query (Latching) State Button
`ssb <n> state`	Set (Latching) State Button
`sta <index> "text"`	Append to Scrolling Textbox
`stc <index>`	Clear Scrolling Textbox
`std <index> <type> <x0> <y0> <x1> <y1>`	Define Scrolling Textbox
`stp <index> "text"`	Prepend to Scrolling Textbox
`std <index>`	Query Scrolling Textbox
`su`	Screen Unblank (Basic)
`SU`	Screen Unblank (Complete)
`sv idx val`	Slider Value
`t "text string" x y [R|T|X|TR|N]`	Text Display
`t "text string"`	
`t? "text string"`	Get Text Display Width In Pixels
`ta [L|C|R][T|C|B]`	Set Text Alignment
`tc`	Touch Calibrate♦
`temp`	Read Temperature
`tf <index> [t] "text string" x y [R|T|X|TR]`	Text Flashing Display
`tf? <n>`	Text Flashing List
`tfc`	Text Flashing Clear
`tfd <index> <state>`	Text Flashing Disable
`tfe <index>`	Text Flashing Enable
`tfs`	Text Flashing Synchronize
`tfx <index>`	Text Flashing Delete
`tm [R|T|X|TR|N]`	Set Text Mode
`touch [on/off]`	Touchscreen On/Off

`tr x0 y0 x1 y1 x2 y2 [color]`	Draw Triangle
`typematic[s] <delay> <repeat>`	Set Typematic Parameters♦
`utf8 [on|off]`	UTF8 Enable / Disable
`v <on|off>`	Display On/Off
`vers`	Version
`w <number of milliseconds>`	Wait
`wr x y [index]`	Window Restore
`wrf <x> <y>`	Wait for Refresh
`wrr x y <width> <height> <index> [<address>]`	Window Restore Rectangle
`ws x0 y0 x1 y1 [index]`	Window Save
`wvr [<line>] [<line2>]`	Wait Vertical Retrace
`x <n> x0 y0 x1 y1`	Define Hotspot
`xa[q] <n> action <args>`	Touch Macro Assign w/ Parameters
`xbb [+|-]<level>`	External Backlight Brightness♦
`xbbs[f] [+|-]<level>`	External Backlight Brightness
`xbl <on|off>`	External Backlight On/Off
`xc <n>`	Clear Hotspot
`xc all`	Clear All Hotspot
`xd <n | "all"> [<n last>]`	Disable Touch
`xe[s] <n | "all"> [<n last>]`	Enable Touch
`xi <index> x y`	Display Bitmap Image
`xic <index> x y x0 y0 x1 y1`	Display Clipped Bitmap Image
`xim <index> x y`	Display Bitmap Image Centered
`xio <bitmap index> <x> <y> <0|1> <length> <offset>`	Display Windowed Bitmap Image
`xm <touch index><macro index | name> [<macro2 index | name>]`	Touch Macro Assign
`xma`	External Memory Available
`xmc FEEB`	External Memory Chip Erase
`xme <addr> FEEB`	External Memory Block Erase
`xmq <touch index><macro index> [<macro2 index>]`	Touch Macro Assign Quiet
`XR <hex register>`	Read LCD Controller
`xs <n> x0 y0 x1 y1`	Define Special Hotspot (Invisible)
`xsnb <n> x0 y0 x1 y1`	Define Special Hotspot (Invisible, No Beep)
`xset <n> [+|-][p|r|t|T|x]`	Set Touch Characteristics
`xst <n> x0 y0 x1 y1`	Define Special Typematic Touch
`xt <n> x0 y0 x1 y1`	Define Typematic Touch Area
`XW <hex register> <hex value>`	Write LCD Controller
`xxy <n> x0 y0 x1 y1`	Define Relative X-Y Hotspot
`xxynb <n> x0 y0 x1 y1`	Define Relative X-Y Hotspot (Silent – No Beep)

`y <milliseconds> | stop`	Animation Yield
`z`	Clear Screen
`zs`	Clear Screen Special

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.

ALLOW PRE_EXISTING TOUCH

Description: Allows buttons and hotspots that are defined where touch is currently active to react immediately. This is useful in cases where an existing screen with hotspots is replaced with a new screen and hotspots and the new screen expects to see the touch release. Without this setting, the new hotspot is ignored until the existing touch is removed.

Command: `*apt`

Arguments: None

Example: `*apt`

ALLOW PRE_EXISTING TOUCH, NO BEEP

Description: Same as `*apt`, but new hotspot does not beep.

Command: `*aptnb`

Arguments: None

Example: `*aptnb`

ANIMATION CLEAR (TEXT FLASHING CLEAR)

Description: Clears the animation and flashing text definitions and disables the animation engine.

Command: `anic` or
`tfc`

Arguments: None

Example: `anic`
Clears the animation buffers and stops the animation engine.

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 ANIMATION YIELD) is used to suspend the animation for a specified period of milliseconds. The animation may be suspended at any "Yield" point (see ANIMATION DISABLE).

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.

An animation executes in a temporary state (see State Save), which exists only until the animation yields. Properties like foreground color must be set within the yield point they are to be used.

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

Arguments: `<n>` The index of the animation, 0 through 9

`<text string>` Any valid command ***Except:***

- An animation Define / Flashing Text Define command.
- A wait command (use Yield instead)
- A State Save/Restore command

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 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 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 #]	Optional, a reference to one of an animation's yield commands.

Yields are numbered for each animation starting at 0 (zero); thus, if an animation contains 4 yields, they are numbered 0--3. If [Yield #] is not given, or is not a valid number, the animation will stop at the next yield it encounters.

Note: Animations are "stopped" by jumping to and executing the commands that precede the 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 LIST (TEXT FLASING 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.
NOTE: the TEXT FLASHING DISPLAY command uses an animation.

Command: `ani? <n>` or
`tf? <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 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.

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.

APPEND TO SCROLLING TEXTBOX (SLCD43 only)

Description:	Appends a line to the bottom of an existing Scrolling Textbox. If the textbox is full, existing lines are scrolled up and the top-most line is discarded.
Command:	`sta <index> "text"`
Arguments:	`<index>` Index of scrolling textbox. `"text"` Text to append, in double-quotes.
Example:	`sta 0 "Sample text"`

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>]` Set beep frequency, and make it "permanent"
Command:	`bfs [<hertz>]` Temporary beep frequency change.
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 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. The maximum count value is 65535.

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. The maximum value is 65535.

<off> is number of ms to stay silent before beeping again. The maximum value is 65535.

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

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 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. The maximum value is 65535.

BINARY BMP DOWNLOAD

Description:	Enables a raw binary data stream to be written to a rectangular area on the SLCDx screen.
Command	`bbmp <index> x0 y0 x1 y1 [timeout]`
Arguments:	`index` – reserved for future use – must be set to zero. x0, y0, x1, y1 – these coordinates define the target screen area `timeout` – optional parameter, 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. If timeout is not specified, the default value is 2000ms (2 seconds).
Notes:	1. If the command is accepted, the SLCDx 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 SLCDx to the host *MUST* be obeyed. No more than 64 characters may be sent after an XOFF (0x13) is received by the host.

BINARY CHART VALUES

Description: Same as CHART VALUES except that multiple values are sent using binary data format for faster drawing speed. Use this command if the standard chart values command is too slow. Note that the pen value is limited to maximum 65535. There is a timeout of 5 seconds if the amount of data specified in the command is not received. If a timeout occurs, the error prompt `"!\r"` is returned.

Command:
```
bcv n number_of_pen_values_to_follow
<lo byte pen0 value><high byte pen0 value>
<lo byte pen1 value><high byte pen1 value>
.     .     .
<lo byte penN value><high byte penN value>
.     .     .
```

Arguments:

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

`number_of_pen_values_to_follow` - this is equivalent to the number of bytes to follow divided by two (pen values are 16-bit), divided by the number of pens. Pen values are interleaved when multiple pens are used.

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

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

Use a pen_value of 0xFFFF as a "no value" placeholder to skip points.

Example:
Send: `"cd 0 10 20 110 120 1 4 0 99 333 2 0FF 1 F00\r"`
Get: `">\r"`
Send: `"bcv 0 5\r"`
Get: `">\r"`
Send (20 bytes): `0x00,0x00, 0x00,0x00, 0x05,0x00, 0x0a,0x00, 0x0a,0x00, 0x14,0x00, 0x0f,0x00, 0x1e,0x00, 0x14,0x00, 0x28,0x00`
Get: `">\r"`

The example defines a chart (see CHART DEFINE) and adds five values to each pen: 0,5,10,15,20 for the teal pen, and 0,10,20,30,40 for the red pen. Each data point is 4 pixels to the right of the last one. Note that the example above is in ASCII with commas between values for readability – the raw data would be in binary without the leading "0x" and without the commas and spaces.

BINARY DOWNLOAD

Description: Enables a raw binary data stream to be written to the SLCDx 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 4, 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 Flash memory.

`offset` – offset from beginning of selected memory area to store pixel data. *NOTE: value is interpreted as HEX, whether or not preceded by 0x*

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

BINARY NOTIFICATION MODE

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

Due to parsing constraints, it is sometimes useful to have the SLCDx 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<index_Byte>
x<index> <Xr> <Yr><return>	Y<index_Byte><Xr_LSByte><Xr_MSByte> <Yr_LSByte><Yr_MSByte>
r<index><return>	R<index_Byte>
s<index><state><return>	S<index_Byte><state_Byte>

l<index><value><return>	L<index_Byte><value_LSByte><value_MSByte>
m<index><return>	M<index_Byte>
e<index><return>	E<index_Byte>

Returns: `on<return>`

or

`off<return>`

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.

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. Button text is limited to no more than 19 characters for button numbers less than 118 and 49 characters for button numbers 118 and up (SLCD+ is always limited to 19 characters).
`"text1"`	Additional argument for BUTTON DEFINE – LATCHING STATE types; text displayed on button in pressed state. The current foreground color will be used for the text. Button text is limited to no more than 19 characters for button numbers less than 118 and 49 characters for button numbers 118 and up (SLCD+ is always limited to 19 characters).
`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.

Example 1: `bdc 23 150 100 1 "Test" 10 11`

Defines button number 23 displayed at x=150, y=100. The "un-pressed" image uses bitmap 10 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 11 is used.

Host notification: See BUTTON DEFINE – MOMENTARY command.

Example 2: `bdc 24 150 200 2 "ON" "off" 10 11`

Defines button number 24 displayed at x=150, y=200. The "un-pressed" image uses bitmap 10 with the text "ON" centered on it. The "pressed" image uses bitmap 11 with the text "off" centered on it.

Host notification: See BUTTON DEFINE – LATCHING STATE command.

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. Initial state is set to state 0 (not pressed).
- 20 Latching. Same as above. Initial state is set to state 0 (not pressed).
- 21 Latching. Same as above, but with initial state set to state 1 (pressed).

`"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. Button text is limited to no more than 19 characters for button numbers less than 118 and 49 characters for button numbers 118 and up (SLCD+ is always limited to 19 characters).

`"text1"` Text string to be displayed on the button in state 1. The current foreground color will be used for the text. Button text is limited to no more than 19 characters for button numbers less than 118 and 49 characters for button numbers 118 and up (SLCD+ is always limited to 19 characters).

`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 – 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 the 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. Button text is limited to no more than 19 characters for button numbers less than 118 and 49 characters for button numbers 118 and up (SLCD+ is always limited to 19 characters).

`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>`

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>`

CHART CLEAR DISPLAY AREA

Description: Clears a given chart’s display area and resets its pens to their starting position.

Command: `cc <n>`

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

CHART DEFINE

Description: Creates a chart to which data can be added. See the CHART VALUES 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): initially, data is added at the left edge of the chart until the right edge is reached; then, 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.
- 3 (STRIP starting at RIGHT EDGE): data is always added at the right edge of the chart after shifting the current data left.
- *NOTE: type 2 is reserved for internal use by the "cdb" command (see CHART DEFINE with BITMAP)*

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, 1, or 3 (see Description, above)

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

`bv` - bottom data value (lowest y value) , positive numbers only

`tv` - top data value (highest y value) , positive numbers only

`bc` - background color in RGB444 format (see COLOR SPECIFICATIONS)

`<pens>` - one or more sets of two values: pen width and pen color; up to 8 pens can be defined. Width can be 1 -- 5, 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 DEFINE with BITMAP

Description: Creates a chart to which data can be added. See the CHART VALUES 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; up to 8 pens can be defined. Width can be 1 -- 5, color is in RGB444 format (see COLOR SPECIFICATIONS).

Example: `cdb 0 10 20 4 0 99 72 2 0FF 1 F00`

Defines a chart in the rectangular area defined by bitmap index 72, starting in the upper left (10,20). The lower right is defined by the bitmap's width and height. Each data value will be 4 horizontal pixels wide. The chart ('Y') values are scaled from 0 to 99. The background bitmap is index 72. Two pens are defined: the first is pen width 2, color teal (0FF), the second is pen width 1, color red (F00).

CHART REDEFINE BACKGROUND BITMAP

Description: Redefines a given chart's background bitmap; useful to highlight a portion of a trace on a chart with a bitmap background.

Command: `cdbb <n> <bitmap>`

Arguments:

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

`bitmap` – bitmap index

CHART REDEFINE BACKGROUND COLOR

Description: Redefines a given chart's background color; useful to highlight a portion of a trace.

Command: `cdbc <n> <color>`

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

`color` – color in RGB444 format (see COLOR SPECIFICATIONS)

CHART REDEFINE CLEAR-BEFORE-DRAW MODE

Description: Redefines the Clear Before Draw Mode of a given OSCILLOSCOPE type chart (with or without a bitmap background). The default mode is on, which causes the chart to clear the area before drawing a new value.

Command: `cddm <n> <on|off>`

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

CHART REDEFINE PEN

Description: Redefines a given chart's pen width and pen color.

Command: `cdp <n> <pen> <width> <color>`

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

`pen` – pen number, 1-8

`width` – pen width

`color` – color in RGB444 format (see COLOR SPECIFICATIONS)

CHART REDEFINE RETRACE MODE

Description: Redefines the Retrace Mode of a given OSCILLOSCOPE type chart (with or without a bitmap background). The default mode is on, which causes the chart pens to reset to the start of the chart when the chart is full. Selecting off causes the chart to stop updating when the chart is full.

Command: `cdrm <n> <on|off>`

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

CHART RESET PENS TO START

Description: Resets a given chart's pens to their starting position, without changing what's already displayed.

Command: `cr <n>`

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

CHART VALUES

Description: Adds data points to a 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. A value or a minus sign must be supplied for each pen defined. If a minus sign character, '-', is present instead of a pen_value, the associated pen will not draw on the chart, leaving a gap in the line for that pen.

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
cv 0 - 60
cv 0 40 -
```

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. The third cv command does not draw the teal pen, but does extend the red pen another 4 pixels. The fourth cv command draws a 4 pixel line segment at value 40 with the teal pen, but does not draw the red pen.

CHECK COLOR MODE

Description: Check if the controller firmware is 8 bit or highcolor.

Command: `*IS16`

Returns:

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

CLEAR ALL HOTSPOT

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

Command: `xc all`

CLEAR HOTSPOT

Description: Clears the previously defined hotspot touch area.

Command: `xc <n>`

Arguments: <n> - Hotspot index (128-225).

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 the '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.

CLEAR SCROLLING TEXTBOX (SLCD43 only)

Description: Clears an existing Scrolling Textbox.

Command: `stc <index>`

Arguments: `<index>` Index of scrolling textbox.

Example: `stc 0`

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`

CONTROL PORT AUTOSWITCH

Description: The SLCDx has 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 SET AUX ESCAPE).

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.*

Command: `*CEXT [<from> <to>]`

Arguments: `[<from> <to>]` optional start and stop offsets used to specify a section of the external flash to CRC. Values are interpreted as HEX numbers. Valid ranges are 0x0 to (external memory available - 1); see EXTERNAL MEMORY AVAILABLE .

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.

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.

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*>

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*>

DEFINE DISPLAYABLE CURSOR (SLCD+ or SLCD6 only)

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. **(Not suitable for new applications – see *simtouch instead.)**

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. Bitmap must be 8 bits per pixel, indexed.
2. Only first 3 colors in color table are used.
3. 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.
4. Maximum cursor size is 64x64.
5. Bitmap index value 0 is used to clear the cursor.

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 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 RELATIVE X-Y HOTSPOT

Description: Same as DEFINE SPECIAL HOTSPOT except that the notice returned when touched will include the x and y coordinates of the touch, relative to the top left corner of the area, as defined by x0 and y0. Also, a touch will set internal variables Xa and Ya to the absolute x and y coordinates of the touch, and will set Xr and Yr to the relative coordinates.

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

Arguments, Returns: same as DEFINE HOTSPOT SPECIAL command.

Example: `xxy 135 100 100 179 139`

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

Example notification when hotspot is pressed at screen coordinates x=105, y=110:

`x135 5 10<return>`

DEFINE RELATIVE X-Y HOTSPOT (Silent – No Beep)

Description: Same as DEFINE RELATIVE X-Y HOTSPOT but without the beep.

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

DEFINE SCROLLING TEXTBOX (SLCD43 only)

Description: Define a new Scrolling Textbox, using the current font, foreground color, and background color. Note that scrolling textboxes require monospaced fonts. See Monospaced Fonts for a list of built-in monospaced fonts.

Command: `std <index> <type> <x0> <y0< <x1> <y1>`

Arguments:

`<index>`	–	Index of scrolling textbox. Valid values are 0-9.
`<type>`	–	Must be 0. Simple terminal-like display, no line buffering.
`<x0> <y0>`	–	top/left corner of textbox location
`<x1> <y1>`	–	bottom/right corner of textbox location

Example: `std 0 0 1 80 165 180`

Defines a scrolling textbox at coordinates 1/80 165/180.

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`

Defines a rectangular hotspot with height of width of 80 and height of 40, located at 100,100.

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

DEFINE SPECIAL HOTSPOT (Invisible, No Beep)

Description: Same as DEFINE HOTSPOT except that the touch area is not reverse video highlighted and there is no audible beep when touched. This allows a silent, hidden touch area to be placed on the screen.

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

Arguments,Returns: same as DEFINE HOTSPOT command.

DEFINE SPECIAL TYPEMATIC TOUCH AREA

Description: Same as DEFINE TYPEMATIC TOUCH AREA except that the touch area is not reverse video highlighted when touched.

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

Arguments, Returns: same as DEFINE HOTSPOT command.

DEFINE TOUCH ACTION

Description: The touch screen can have different operating characteristics defined using this command. It stores the value in non-volatile memory (see warning); when given no arguments, returns the current setting. If the optional ‘s’ is present, does NOT store the value.

Command: `*touchMode[s] [S|L|W]`

Arguments:
(none) - display current settings
`S` - standard (non of the others).
`L` - lockout other touch areas until a touch release (turn off "n-key rollover) .
`W` - wandering from the initial press will cause the touch to release. Note: selecting W will also select the L setting.

Example: `*touchMode LW`

This sets the touch action to be the most guarded against unintentional presses at the cost of less responsive feel.

Example: `*touchMode`

Returns: `-LW`

DEFINE TOUCH PARAMETERS

Description: The touch screen can have different sensitivities defined using this command. These values are dependent on the touch panel used.

Command: `*touchParm [<samples> <span>]`

Arguments:
`(none)` – display current settings
`<samples>` – number of touch samples required for a valid touch; the larger the number the less sensitive.
`<span>` – allowable range for sample location measurement; the smaller the number the less sensitive.

Example: `*touchParm 8 12`

DEFINE TYPEMATIC TOUCH AREA

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

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

Arguments, Returns: same as DEFINE HOTSPOT command.

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`

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 | "all"> [<n last>]`

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

"all" disables all touch areas or buttons (entire screen).

<n last> optional parameter that is the highest number of touch area or button; if present, all touch areas and buttons are disabled incrementally from <n> thru <n last>.

Example: `xd 1`

Disables previously defined button 1.

xd 2 10

Disables touch area or buttons numbered 2-10.

DISPLAY BITMAP IMAGE

Description: Copies a 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 SLCDx 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 BITMAP IMAGE CENTERED

Description: Same as xi command above, except the bitmap is centered at (x, y).

Command: `xim <index> x y`

Arguments: `<index>` – bitmap index.

`x y` - location of center of bitmap.

Example `xim 4 100 120`

This displays the 4th bitmap centered at location (100, 120).

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. (bitmap should not be compressed by BMPLoad)

`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 OEM BITMAP IMAGE

Description:	Copies a factory programmed bitmap onto the screen at x y (top left corner of bitmap target). It returns syntax error if the 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

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>`

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 . **Only uncompressed bitmaps are supported by this command.**

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. (bitmap should not be compressed by BMPLoad)

`<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

Note: Highcolor firmware will only support Highcolor 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.

DRAW ARC SEGMENT

Description: Draws an arc segment using the current foreground color and pen width.

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)

Angles are CCW from 0=horizontal right (on a clock, 3:00, on a compass, East). <End angle> must be greater than <Start angle>.

Example: `a 100 100 40 20 110`

Draws a semi-circle centered at 100X100.

DRAW CIRCLE

Description: Draws a single pixel width circle using the current 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 is 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 ELLIPSE

Description: Draws a single pixel width ellipse using the current foreground color.

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 oriented.

DRAW FILLED ELLIPSE

Description: Draws a filled ellipse using the current foreground color.

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 an ellipse centered at 150X150, Ellipse is vertically orientated.

DRAW FILLED POLYGON

Description: Draws a filled polygon at specified origin, using current foreground color.

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 0 0 15 15 15 10 40 10 40 -10 15 -10 15 -15 0 0`

Draws a filled polygon at offset 100 100.

DRAW LINE

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

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 OUTLINE POLYGON

Description: Draws a polygon at specified origin, using the current foreground color and pen width.

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 0 0 15 15 15 10 40 10 40 -10 15 -10 15 -15 0 0`

Draws a polygon at offset 100 100.

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 POLYLINE

Description: Draws a polyline at the specified origin, using the current foreground color and pen width. A polyline is a line with multiple segments that may not connect to form a complete shape.

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 35 -15 0 0 35 15`

Draws polyline at offset 100 100.

DRAW RECTANGLE

Description: Draws a rectangle using the current foreground color and pen width, or an alternate style and color

Command:

```
r x0 y0 x1 y1

r x0 y0 x1 y1 <style>

r x0 y0 x1 y1 1 [color]
```

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

<style>: 1=filled, 2= one pixel wide dotted line.

[color]: fill color in RGB444 format (see COLOR SPECIFICATIONS); if not present, uses foreground color.

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 130 and a height of 20.

DRAW ROTATED FILLED POLYGON

Description: Draws a rotated, filled, polygon at the specified origin, using the current foreground color.

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 0 0 15 15 15 10 40 10 40 -10 15
-10 15 -15 0 0
```

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

DRAW ROTATED POLYGON

Description: Draws a rotated polygon at the specified origin, using the current foreground color and pen width.

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 0 0 15 15 15 10 40 10 40 -10 15
-10 15 -15 0 0
```

Draws polygon rotated 45 Deg. at offset 100 100.

DRAW ROTATED POLYLINE

Description: Draws a rotated polyline at the specified origin, using the current foreground color and pen width.

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 35 -15 0 0 35 15
```

Draws polyline rotated 45 Deg. at offset 100 100.

DRAW TRIANGLE

Description:	Draws a triangle using the current pen width and foreground color for the line. If the optional fill color argument is supplied, the triangle is also filled with the specified color.
Command:	`tr x0 y0 x1 y1 x2 y2` [color]
Arguments:	The three x, y sets are the triangle vertices. The optional color fill argument is in RGB444 format (see COLOR SPECIFICATIONS).
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.

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. When writing values, be aware the EEPROM chip has limited total write cycles (see the warning).
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>`

ENABLE TOUCH

Description: Re-enables touch area or button. For buttons, the state of the button is remembered and the correct graphic is displayed. If the optional [s] variation is used, the button bitmaps are not shown on re-enable.

Command: `xe[s] <n | "all"> [<n last>]`

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

"all" re-enables all touch areas or buttons (entire screen).

<n last> optional parameter that is the highest number of touch area or button; if present, all touch areas and buttons are re-enabled incrementally from <n> thru <n last>.

Example: `xe 1`

Enables previously disabled button 1.

xe 2 10

Enables touch area or buttons numbered 2-10.

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 command.

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

Arguments: <level> is a 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. Using the [sf] option overrides the internal brightness limit and allows the screen to be turned down to its lowest settings.

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).

EXTERNAL BACKLIGHT ON/OFF

Description: Turns the external backlight control on or off via J10 on the SLCD+/SLCD6 or via J6 on the SLCD43.

Command: `xbl <on|off>`

EXTERNAL MEMORY AVAILABLE

Description: Returns the size of External Flash (macro/bitmap) memory, in bytes.

Command: `xma`

EXTERNAL MEMORY BLOCK ERASE

Description: Erases a 64KB block of External Flash (macro/bitmap) memory starting at the given Hex address.

Command: `xme <addr> FEEB`

Note: May take as long as 10 seconds to complete.

EXTERNAL MEMORY CHIP ERASE

Description: Erases all of External Flash (macro/bitmap) memory.

Command: `xmc FEEB`

Note: May take as long as 60 seconds to complete.

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`

GET TEXT DISPLAY WIDTH IN PIXELS

Description: Returns the width of the space required to display the given text using the current font, in pixels.

Command: `t? "text string"`

Examples:

```
f 20B
>
t? "string one"
134
>
t? "string TWO"
149
>
```

GET VARIABLE

Description: Used to return the value of an internal variable (Integer, EEPROM, String, and Point Coordinate).

Command: `get <internal variable name>`

Arguments: `<internal variable name>`

```
Integer             – i0 thru i19
EEPROM val          – e0 thru eF
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 VARIABLE (HEX)

Description: Used to return the HEX value of an internal variable (Integer or EEPROM only).

Command: `getx <internal variable name>`

Arguments: `<internal variable name>`

```
Integer             – i0 thru i19
EEPROM val          – e0 thru eF
```

Returns: `<HEX value><return>`

Example:

```
get i9
A5
```

Internal Integer variable i9 returns its value, HEX A5.

I2C READ (SLCD43 only)

Description: Reads up to 10 bytes of data from the device at <addr>, starting with register <reg>. The <addr> and <reg> parameters are assumed to be HEX values, and the <length> value is assumed to be in decimal. Prints results in HEX.

Command: `i2cr <addr> <reg> <length>`

Arguments: `<addr>` – Slave/device address, shifted left 1 bit to account for read/write bit. Firmware will set bit zero appropriately.

`<reg>` – Register address within device to start read operation.

`<length>` – Number of bytes to read.

Example: `i2cr a0 70 10`

Returns: Data such as: `11 22 33 44 55 66 77 88 99 AA`

I2C SPEED (SLCD43 only)

Description: Sets the I2C bus speed. Options are 100 or 400, for 100khz and 400khz respectively. The system default is 100khz, and it is assumed that the user of this command will restore the default speed of 100khz after completing operations at higher speeds.

Command: `i2c speed <speed>`

Arguments: `<speed>` – Either 100 or 400.

Example: `i2c speed 400`
Immediately changes the I2C bus speed to 400khz.

I2C WRITE (SLCD43 only)

Description: Write a single byte of data to the device at <addr>, register <reg>. All parameters are assumed to be HEX values.

Command: `i2cw <addr> <reg> <data>`

Arguments: `<addr>` – Slave/device address, shifted left 1 bit to account for read/write bit. Firmware will set bit zero appropriately.

`<reg>` – Register address within device to start read operation.

`<data>` – Data to write.

Example: `i2cw a0 70 3A`
Writes the value 3A to register 70 in the device at (shifted) address A0.

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 LEVELBAR VALUE).

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, x1,y1`	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; negative numbers allowed
`bc`	Background color in RGB444 format (see COLOR SPECIFICATIONS)
`<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.

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 [index]`

Arguments: `[index]` optional bitmap index number

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 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 [index]`

Arguments: `[index]` optional macro index number

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: :aborting on *abt<return> … 1 line per macro call level

'>'<return>	The success prompt, indicates successful abort of executing macro(s).

Example: `*abt`

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

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 the maximum size of arguments is only limited by total command line length (max 128).

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.

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>`

MEMORY POP

Description: Removes one or more characters from the end of a string variable. If [index] is 0-9, removes <number> of characters specified from the end of the corresponding string var, s0-s9; otherwise, removes from the end of memory variable `M`. If <number> is -1, all characters are removed.

Command: `mpop [index] <number>`

Arguments:
`index` – selects string variable s0 through s9
`number` – number of characters to remove from string variable

Example: `mpop 1 3`

This removes 3 characters from the end of the `s1` variable.

MEMORY PUSH

Description: Appends a string to a string variable. If [index] is 0-9, <string> will be appended to the corresponding string var, s0-s9; otherwise <string> will be appended to memory variable `M`. Length is limited to 80, or [max] if given (must be between 1 and 80).

Command: `mpush [index] "<string>" [max]`

Arguments:

`index`	– selects string variable s0 through s9
`string`	– string to append to contents of string variable
`max`	– maximum length of string variable

Example: `mpush 1 "0" 3`

This appends "0" to `s1` variable, unless length is already 3.

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. The needle is drawn using the current foreground color. **Only uncompressed bitmaps are supported by this command.**

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`	– angle of minimum numerical value for indicator.
`maxAngle`	– angle of 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 degrees is as drawn and degrees (only positive) move clockwise.

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

The minVal, maxVal, Init_val, minAngle and maxAngle parameters are all unsigned integers, so negative numbers are not allowed. If your meter needs to display negative numbers, the application will need to bias the negative values so the meter sees only positive numbers.

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 DEFINE BAND

Description: Creates a "Meter" object that resembles an analog meter (with an indicator) with a settable color band. The meter object uses a background bitmap that visually represents the meter, a polygon for the indicator needle, and arcs for the band. The needle is drawn using the current foreground color.

Command:

```
mdb <ix> <bitmap> <x> <y> <type> <minVal>
<maxVal> <init_val> <minAngle> <maxAngle>
<bandMin> <bandMax> <bandRadius> <bandPen >
<bandColor> <bandBG> <pivotX> <pivotY> <x1, y1
... [x10, y10]>
```

Arguments:	`ix`	– meter index. The meter index must be in the range 0 to 7 (maximum 8 meters).
	`bitmap`	– background bitmap index (bitmap should not be compressed by BMPLoad).
	`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.
	`bandMin`	– minimum/lowest value in band range.
	`bandMax`	– maximum/highest value in band range.
	`bandRadius`	– radius of arc used to draw band.
	`bandPen`	– width of pen used to draw band.
	`bandColor`	– foreground color of band in RGB444 format (see COLOR SPECIFICATIONS).
	`bandBG`	– background color of band in RGB444 format (see COLOR SPECIFICATIONS).
	`pivotX, pivotY`	– 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 degrees is as drawn and degrees (only positive) move indicator clockwise.

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

Example:

```
mdb 0 3 282 150 1 0 8000 0 60 300 0 3000 105 5
0f0 ccc 120 115 -5 0 0 85 5 0
```

This example defines a meter with band, with index number 0, using bitmap index 3 as the background image, located at X/Y of 282/150. The type is always 1. The minimum indicator value of 0 is at angle 60 degrees (clockwise from vertical bottom), and the maximum value of 8000 is at angle 300 degrees (clockwise from vertical bottom). The indicator will point to initial value 0. The band range is from 0 to 3000, the band radius is 105 degrees, the band pen width is 5 pixels, the band foreground color is green, and the band background color is grey. The indicator pivot point is 120/115, and the indicator is drawn as a triangle with polygon points -5 0 0 85 5 0.

METER VALUE

Description: Sets the value of an ID 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

METER VALUE BAND

Description: Sets the value of the indicator for a specified meter. The meter must have been previously created by the Meter Define Band command. The second form of the command allows the use of parameters from a previous full command, and is useful if none of the other parameters are changing for the new indicator value.

Command:
```
mvb <ix> <val> <bandMin> <bandMax> <bandRadius>
<bandPen> <bandColor> <indicatorColor>

mvb <ix> <val>
```

Arguments:

`ix`	– index value for meter with band previously defined.
`val`	– value to set indicator. Must be in the range of values as defined by the Meter Define Band command.
`bandMin`	– lower value of meter band. Must be in the range of band values as defined by the Meter Define Band command.
`bandMax`	– upper value of meter band. Must be in the range of band values as defined by the Meter Define Band command.
`bandRadius`	– radius of meter band arc.
`bandPen`	– pen width of meter band.
`bandColor`	– color of meter band in RGB444 format (see COLOR SPECIFICATIONS).
`indicatorColor`	– color of meter indicator in RGB444 format (see COLOR SPECIFICATIONS).

Example:
```
mvb 0 5400 4000 7000 105 5 ff0 f00
```

This example sets the indicator of meter 0 to the value of 5400, sets the band range of the meter to 4000-7000, sets the radius of the band arc to 105 degrees, sets the band pen width to 5 pixels, the band color to yellow and the indicator to red.

OUTPUT STRING (AUX)

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.

Command:
```
aout "<text string>"
```

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>
```

PIXEL READ

Description: In 8-bit color mode, returns the 2 character hex pixel palette index value for the specified pixel. In high color mode, returns the pixel color in RGB565 format (see COLOR SPECIFICATIONS).

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`

PIXEL WRITE

Description — Sets a single pixel to either the foreground color, or if specified, a specific RGB565 value (see COLOR SPECIFICATIONS).

Command: `pw x y [RGB565]`

Arguments: `x y` – location of pixel

`RGB565` - if not provided, the foreground color is used.

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
```

PREPEND SCROLLING TEXTBOX (SLCD43 only)

Description: Prepends a line to the top of an existing Scrolling Textbox. If the textbox is full, existing lines are scrolled down and the bottom-most line is discarded.

Command: `stp <index> "text"`

Arguments: `<index>` Index of scrolling textbox.

`"text"` Text to prepend, in double-quotes.

Example: `stp 0 "Sample text"`

QUERY CONTROL PORT

Description: Query the control port settings, including the EEPROM setting, the current main port, and the aux port. Returns a string in the form "rN mN aN", indicating the EEPROM main setting, the current main setting, and the current aux setting.

Command: `*com?`

Example: `*com?`

Assuming that the main port is set to COM3, returns "r0 m3 a0".

QUERY SCROLLING TEXTBOX (SLCD43 only)

Description: Query an existing Scrolling Textbox for the number of lines and characters that fit into the display region.

Command: `std <index>`

Arguments: `<index>` Index of scrolling textbox. Valid values are 0-9.

Example: `std 0`

Returns 5 16 0 0, based on a scrolling textbox created at 1/80 165/180 with the font set to M12. The two zero values are reserved for future options.

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.

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. Outputs a ':', followed by the received data in ASCII printable range 0x20-0x7E, followed by a standard prompt 0x3e 0x0d (">"). If optional 'b' is present, outputs in binary, 1st byte output is number of bytes received, followed by the received bytes, then a standard prompt.
Command:	`ain<0-3> [b]`
Argument:	Port as shown above
Example:	`ain3`
Returns:	`:<received data from COM 3><prompt>`

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 TEMPERATURE

Description:	Displays temperature in degrees Centigrade measured by the sensor (LM62 on SLCD+/SLCD6 in location U3, or LM61 on SLCD43 in location U11).
Command:	`temp`
Returns:	`NN.N<return >`
	Where NN.N is the temperature in degrees centigrade. If less than 10, a leading zero is inserted.

REDRAW ROTATED POLYGON

Description:	Redraws a rotated polygon or a rotated filled polygon at specified origin, using the current pen width and foreground color.
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.

RESET BOARD / 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 RESET BOARD/SOFTWARE). 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 that this does not reset the beep frequency to the manufactured state which was calibrated for maximum volume (resonance).

Command: `*MFGRESET`

Returns: "Power on" prompt.

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)

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 call-level has its own memory for state save/restore, including call-level 0 (no macro running); also, the animation engine uses its own memory to save the existing state before animations run, and restoring it after.

Command: sr

Arguments: None

Examples: sr

Restore the drawing state.

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 call-level has its own memory for state save/restore, including call-level 0 (no macro running); also, the animation engine uses its own memory to save the existing state before animations run, and restoring it after.

Command: `ss`

Arguments: None

Examples: `ss`

Save the drawing state.

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 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 RGB444 format (see COLOR SPECIFICATIONS)

Example: `SB 003`

Sets the entire screen to a light blue.

SCREEN UNBLANK (Basic; 8 bit firmware only)

Description: Reverses the effect of the Screen Blank (basic) command

Command: `su`

SCREEN UNBLANK (Complete; 8 bit firmware only)

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

Command: `SU`

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.

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 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[0-3] <baudrate>` or `baud <baudrate>` or `baudp <baudrate>` or `baudTry <baudrate>`
Argument:	`<baudrate>` … valid values are: 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, and 230400.
Example:	`baud0 57600` This sets COM0 to 57600 baud and then outputs the system prompt.
Example:	`baud 19200` This sets the COM port currently configured as the Control/Main port to 19200 baud and then outputs the system prompt.
Example:	`baudp 115200` This outputs the system prompt at the current baud rate, sets the COM port currently configured as the Control/Main port to 115200 baud, and then outputs another system prompt.
Example:	`baudTry 230400` This sets the COM port currently configured as the Control/Main port to 230400 baud and then waits for the string “testing123”. If received within about 5 seconds, outputs the prompt “:ok>”. If the string is not received, sets the baud rate to the previous value and outputs the system error prompt (“!”).
Note:	USB port (COM 3) can be set to 460800 baud.

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. Not applicable for modules using “high-color” firmware.
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 (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 (Detailed)

Description	Sets the background and foreground color for all commands using arbitrary RBG values.
Command	`S <foreground_detail> <background_detail>`
Arguments:	<foreground_detail> = foreground color value in RGB format <background_detail> = background color value in RGB format RGB format = RGB444 or RGB888 format (see COLOR SPECIFICATIONS). Both arguments must be in the same format.
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. When using 8 bit color firmware, the SLCD+ or SLCD6 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. When using high-color firmware, all 4096 colors are usable. 3. This command does not function as described when using a custom palette in 8 bit color mode.

SET CONTROL PORT

Description:	Used to set the port used to control the unit. The value is stored in non-volatile memory and restored on power-on unless the optional 's' is added to the basic command. It also sets the "aux" port to either COM0 or COM1, whichever is not the main port.
Command:	`*com<0-3>main[s]`
Example:	`*com3main` USB serial port is used as the main console.

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
```

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 DISPLAYABLE CURSOR POSITION (SLCD+ or SLCD6 only)

Description: Sets the displayable cursor at an x-y coordinate. **(Not suitable for new applications – see *simtouch instead.)**

Command: `curp x y`

Example: `curp 10 20<return>`

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

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 commands.

x: XOR drawing mode; inverts the existing pixel to draw lines.

Example: `d n`

This sets the drawing mode to normal

SET FONT

Description — Sets the font to be used in subsequent TEXT DISPLAY commands. See the BMPload PROGRAM 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 FONTS.

Example:

```
f 13B
```

Set the current font to 13 point bold.

SET I/O PIN (SLCD43 ONLY)

Description: Sets the selected I/O pin to the specified state.

Command:

```
iopin <pin> <state>
```

Arguments:

`<pin>` – I/O pin number. Pin mappings are shown below.

`<state>` – Use 0 or 1 to specify output state.

I/O Pin #	SLCD43 Physical Output Pin	Signal Name
0	J4 pin 5	P103
1	J4 pin 6	EXPWM
2	J1 pin 2	BUSY

Examples:

```
iopin 0 1
```

Sets I/O pin 0 to 1.

SET or QUERY (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 activated by this command; therefore, when using this command in a macro, the "xd" command should be used to deactivate the button before this command is used, and then the "xe" command should be used afterwards to reactivate the button; otherwise, the macro associated with the button will immediately execute and abort the macro containing the "ssb" command.*

Command: `ssb <n> state`

Arguments: n - latching button number (0-127)

state - specifies the desired state: 0 for not pressed, or 1 for pressed.

Example: `ssb 5 1`

This command would force a button defined with DEFINE BUTTON (type=2) into state 1.

Example: `ssb 5`

This command queries the current state of button 5.

SET LED

Description: Turns the “power LED” on the board on or off. 1 is on and 0 is off. The “power LED” is defined in the various controller board manuals, but is D2 on the SLCD, SLCD+ and SLCD6, and is D3 on the SLCD43 products.

Command: `led [0|1]`

Returns: > (standard prompt)

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 PANEL 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

SET or QUERY PEN WIDTH

Description: Sets the pen width for line drawing commands including line, rectangle *but not circle*. Default setting is width of 2. With no argument, it returns the current setting.

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.

Example: `p`

Returns: `1` (the current pen width is returned)

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`

SET STATE BUTON DELIMITER

Description: Sets the delimiter character used in latching state button host notification. The default is to have no delimiter character in the host notification. This setting is not stored in the display module and must be set after any power cycle or module reset.

Command: `*sdelim 'c'`

Arguments: `c` –character to use as the delimiter. Use 0 (without enclosing single-quotes) to remove the current delimiter.

Example: `*sdelim ':'`

Sets the latching state button delimiter character to a colon. Assuming that a latching state button is defined, such as button 1, the following host notification will be sent when the button is pressed:

`s1:1<return>`

and the following host notification will be sent when the button is pressed again:

`s1:0<return>`

`*sdelim 0`

Clears the latching state button delimiter character. Assuming that a latching state button is defined, such as button 1, the following host notification will be sent when the button is pressed:

`s11<return>`

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: the horizontal alignment reverts to "L" (left) after a text display command is issued, but the vertical alignment remains as set.*** 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 commands. 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".

Note: When used within an animation, only guaranteed to persist until a yield occurs; likewise, within a macro, only guaranteed to persist until an animation interrupts the macro, or another macro starts (via a touch press or release event).

SET TOUCH CHARACTERISTICS

Description: Allows button or hotspot characteristics to be modified after being defined. Useful to make a typematic hotspot.

Command: `xset <n> [+|-][p|r|t|T|x]`

Arguments:

`<n>` index of hotspot or button
`+` (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)

`x` include relative x and y in notification (hotspots only)

NOTE: Only one option may be changed with each `xset` command.

NOTE: A hotspot needs to be a typematic hotspot before the upper-case T works. You can define the hotspot with the 'x' command and then use "xset <n> t" to make it typematic, or you can use the 'xt' command to define it as typematic. Then use “xset <n> +T”.

Example: xset 10 +r

Adds “notify on release” characteristic to button 10.

SET TOUCH 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.

SET TYPEMATIC PARAMETERS

Description: Sets the delay and repeat rate for typematic buttons. The values are stored in non-volatile memory and restored on power-on unless the optional 's' is added to the basic command.

Command: `typematic[s] <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. The default <delay> is 750ms (75) and the default <repeat> rate is 200ms (20).

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 VARIABLE

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

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

Arguments: `<internal variable name>`

`Integer - i0 thru i19`

`EEPROM val - e0 thru eF (see the EEPROM warning)`

`String - s0 thru s9`

`Point Coordinate - p0 thru p9`

Examples: `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".

SET VARIABLE (HEX)

Description: Used to set a HEX value to an internal variable. If an internal variable (Integer or EEPROM only) is used, it should be after this command to have a meaningful value.

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

Arguments: `<internal variable name>`

`Integer - i0 thru i19`

`EEPROM val – e0 thru eF (see the EEPROM warning)`

Examples: `setx i9 A5`

Set internal integer variable i9 to HEX A5.

SIMULATE TOUCH

Description: Simulates a touch at the specified x, y position. Allows a host controller to 'touch' a button or hotspot on the screen. Macros assigned to the button or hotspot are executed as if the screen had been touched.

Command: `*simtouch x y`

Arguments: `x, y` – coordinate within desired button or hotspot

Example: `*simtouch 128 50`

Assuming that a button is defined at that coordinate, such as button 5, the following host notification will be sent:

`x5<return>`

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 255. A maximum of 8 sliders may be defined in this range. 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`	Always one. Value has no impact.
`hi`	maximum slider value, negative numbers allowed
`lo`	minimum slider value, negative numbers allowed

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 slider values range from 100 at the top to 0 at the bottom.

Example notification: `l128:50`

SLIDER VALUE

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

Command: `sv idx val`

Arguments: `idx` - slider index (see SLIDER DEFINE)

`val` - value for the slider.

Example: `sv 128 50`

Sets slider index 128 to value 50.

SPEED TEST

Description:	Command to determine the execution speed of a given command**.**
Command:	`*speedtest N <command line>`
Arguments:	`N` - number of times to run the command. `<command line> -` Command line enclosed in angle brackets. This is done so that quotes (") can be used in the command.
Examples:	`*speedtest 100 <t "Hello World" 0 0>`
Returns:	`:time/command = 4 ms`

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 SLCDxexternal flash memory. See BMPload Program 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.

TEXT DISPLAY

Description: Displays a text string starting at a specified point using the currently set font and text alignment. 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 character block height in the implied box starting at the x pixel location.

Command:

```
t "text string" x y [mode]
```

or

```
t "text string" x0 y0 x1 y1 [[mode][wrap/rotate]]
```

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.

x0 y0 is top left corner of rectangle

x1 y1 is bottom right corner of rectangle

[mode] is one of:

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.

[wrap/rotate] is one of:

WW – wrap text on word boundary

WC – wrap text on char boundary

CW – rotate text 90 degrees clockwise

CCW – rotate text 90 degrees counter-clockwise

I – rotate text 180 degrees (invert)

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

In the first 2 forms of the command (see above) if the text mode (N, R, T, X, TR) is not specified, it will be reset to Normal; if it is specified, it will be used and will persist until another mode is specified (which may occur within an animation or a macro attached to a touch press/release event). Initially, Normal mode is used, and will remain in effect until changed. To prevent confusion about which mode the command will use, always explicitly declare it within the command. This is especially important within macros and animations.

In the second form of the command, there must be no space between the [mode] and the [wrap/rotate] if both are present.

To use any of the rotate specifications, you must use a font that is not anti-aliased.

The XOR mode does not work with anti-aliased fonts.

Examples:

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

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

```
f13B
r 100 100 160 200
ta CC
t "This is in a box" 100 100 160 200 WW
```

displays the text

```
This is in a box
```

centered in a rectangle with word wrap enabled; "This is in" is the 1st line; "a box" is the 2nd line; the rectangle is at 100, 100 is 61 wide and 101 tall.

TEXT FLASHING DELETE

Description: Deletes the specified text flash animation.

Command: `tfx <index>`

Arguments: `<index>` is an identifier for this animation; valid range is 0-9.

Examples: `tfx 0`

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

TEXT FLASHING DISABLE

Description: Disables a flashing text animation as specified by the index (see TEXT FLASHING DISPLAY). The stopping point state can be specified.

Command: `tfd <index> <state>`

Arguments: `<index>` is an identifier for this animation; valid range is 0-9.

<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's animation index, use the "tfd <index> <state>" command to stop the animation at the selected state, and then use the "tfx <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 DISPLAY

Description: Creates an animation to display 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] [mode]`

All parameters except index and "text string" are optional. If unspecified, <t> will default to 500 mS, and [x y] will be the current character position; the default text mode will be Transparent if [x y] is not given or Normal if [x y] is given. A null text string "" is acceptable.

Arguments: `<index>` is an identifier for this animation; valid range is 0-9.

t – the number of milliseconds between flashes.

x – the left edge of the first character areas.

y – the top edge of the first character area.

mode – Text Display Mode (see TEXT DISPLAY)

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`

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; the text will be written in Normal mode, flashing between normal and reverse.

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 an identifier for this animation; valid range is 0-9.

Examples: `tfe 0`

This resumes the text animation from a previously stopped state.

TEXT FLASHING SYNCHRONIZE

Description: Synchronizes all animations.

Command: `tfs`

Arguments: None

Example: `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.

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)

TOUCH DISPLAYABLE CURSOR (SLCD+ or SLCD6 only)

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. **(Not suitable for new applications – see *simtouch instead.)**

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.

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. Use a macro index of 0 to remove the macro assignment.

<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 system-constructed, predefined label (based upon the type of button or hotspot) may be concatenated. The format for these labels only applies to this command. The specific format of the predefined label is related to the host 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'>]

For a latching state button with index 13, the labels would be ":s13_0" (button 13 in state 0) or ":s13_1" (button 13 in state 1).

For a momentary button with index 14 defined to notify on press only, release only, or both, the labels would be ":x14" (button 14 pressed) or ":r14" (button 14 released).

Similarly, for a hotspot with index 150, the label would be ":x150" (hotspot 150 touched or released).

NOTE: when using predefined labels, both macros (pressed and released) must include labels.

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. 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 or released, as specified by <action>, the associated macro is executed with the specified arguments. "`xaq`" is the quiet form of the command which does not generate a host notification, but otherwise behaves the same as "`xa `".

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 the maximum size of each argument, is version-dependent. See MACRO FILES AND FORMAT.

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 with argument `Check` when button 1 is pushed. The corresponding macro definition could look as follows:

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

When button 1 is pushed, a beep sounds, a host notification ":x1" is generated, and macro 17 is invoked with argument "Check"; as defined, macro 17 executes the following command:

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

Example 2: `xaq 1 p 17 Check`

Assuming button 1 and macro 17 have been defined as in the previous example, this causes the 't' command to execute as before, but no host notification will be generated.

TOUCHSCREEN ON/OFF

Description: Enables or disables the whole touch screen, or reports its status.

Command: `touch [on/off]`

Arguments: [on/off] - turns the touch screen on/off; if not given, reports the current setting: "touch on" or "touch off".

Example:

```
touch off
bdc 1  10 10 1 "1"
bdc 2  80 10 1 "2"
bdc 3 150 10 1 "3"
xmq 1 b1_macro
xmq 2 b2_macro
xmq 3 b3_macro
touch on
```

This example causes all touches to be ignored until after the buttons are defined and attached to macros.

UTF8 ENABLE / DISABLE

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.

VERSION

Description: Displays the version and configuration of the firmware.

Command: `vers`

Returns: <version number><matrix addressing>/<color depth>/<orientation><spc><board model><spc>"<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

Example: `2.9.0tft/hc/L SLCD43 "AT043TN24"`

WAIT

Description: Returns a 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.

WAIT FOR REFRESH

Description: Similar to WAIT VERTICAL RETRACE, however it works in any screen orientation. Returns when a vertical (landscape orientation) or horizontal (portrait orientation) display retrace occurs. This command is used to avoid "tearing" or “flashing” in animations.

Command: `wrf <x> <y>`

Arguments:

`<x>`	Utilized in portrait orientation mode. This is the x coordinate to wait for refresh. The number of vertical scan lines to wait is the X screen resolution minus x-line value. Note that the vertical scan lines start from the right to left of screen when viewing in portrait orientation.
`<y>`	Utilized in landscape mode. This is the y coordinate. This is also the number of horizontal scan lines to wait after retrace. Note that the horizontal scan lines start from the top to bottom of screen when viewing in landscape orientation.

Note: For best results, bitmaps for animations should be stored uncompressed in flash memory.

Example:
```
wrf 100 35
```

This example is in portrait orientation mode. This waits until vertical scan lines refresh the screen until reaching x coordinate 100. For a QVGA LCD screen (320x240 pixels), this would wait 220 vertical scan lines (320-100 = 220). Note this is the case since scan lines are updated from right to left when viewing in portrait orientation.

WAIT VERTICAL RETRACE

Description: Returns when a vertical display retrace occurs with an optional offset. Used to avoid "tearing" or “flashing” in animations. This command only works in landscape orientation mode.

Command: `wvr [<line>] [<line2>]`

Arguments:

`<line>`	Optional number of horizontal scan 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 horizontal vertical lines. Useful to put in an animation script before displaying a bitmap at x,100 with height 35.

WINDOW RESTORE (SLCD+ or SLCD6 only)

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 (SLCD+ or SLCD6 only)

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 SLCD43 controller boards.

WINDOW SAVE (SLCD+ or SLCD6 only)

Description: Saves contents of a rectangular screen area to an off-screen buffer. Maximum size available depends on color depth (see tables below).

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.

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>`

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.

3. BMPload PROGRAM

3.1 Overview

The SLCDx 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 SLCDx flash 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 SLCDx 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 SLCDx. 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 SLCDx 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 (see RGB565 in COLOR SPECIFICATIONS). *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 lower-case 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*.

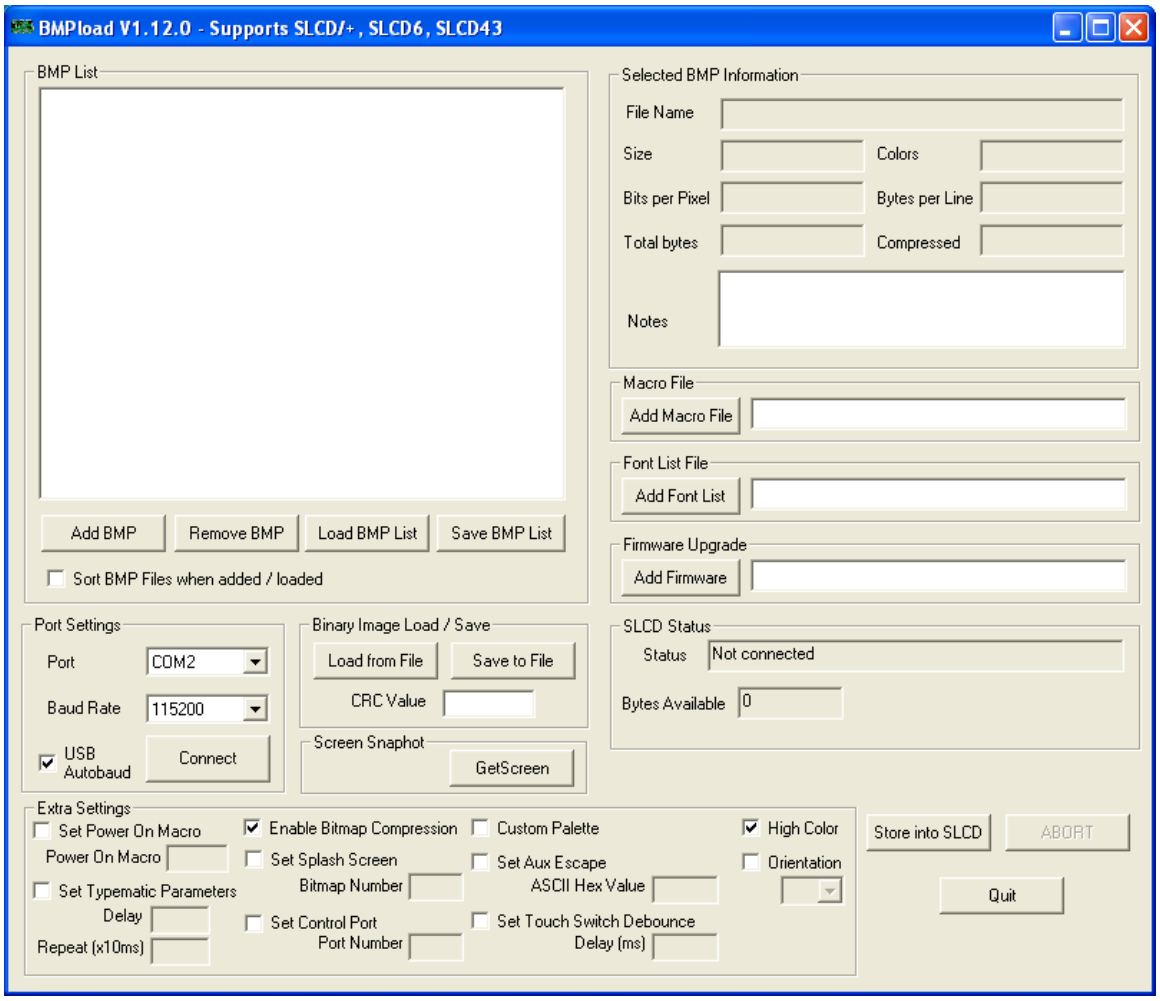
3.5 Bitmap file naming convention

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 Windows 7. In order to download bitmaps and so forth to the board, the computer running BMPload should have a serial port connected to the SLCDx. 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:

BMP List - *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.

BMP List - *Remove BMP*

Remove one or more bitmaps from the BMP List window.

BMP List - *Load BMP List*

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

BMP List - *Save BMP List*

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

BMP List - *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 System Independent Fonts (.SIF) 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.*

Antialiased fonts are supported, but the font filename must contain a special substring for each type. Antialiased fonts which have 2 bits per pixel (bpp) antialiasing information, should have a “aa2” (case insensitive) and 4 bits per pixel (bpp) antialiasing information should have a “aa4” in the filename.

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 SLCDx board.

Port Settings - *Baud Rate*

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

Port Settings - *USB Autobaud*

If the SLCDx 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 SLCDx 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 SLCDx image is programmed via the serial port. It is retrieved from the SLCDx via the *CEXT 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.

Extra Settings Group

Note: As of BMPload version 1.11.2, any unselected Extra Setting will be set to the default value by BMPload.

Extra Settings - *Set Power On Macro*

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

Extra Settings - *Set Typematic Parameters*

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

Extra Settings - *Set Splash Screen*

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

Extra Settings - *Set Control Port*

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

Extra Settings - *Set Aux Escape*

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

Extra Settings - *Set Touch Debounce*

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

Extra Settings - *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 lower-case 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.

Extra Settings - *Custom Palette*

This is only applicable to 8 bit color firmware. The SLCDx 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.

Extra Settings - *High color*

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

Extra Settings - *Orientation*

This control consists of a checkbox, and then a drop-down menu is enabled for

options. See SET ORIENTATION (rotate display 180 degrees) for details.

Store into SLCDx

This starts the serial download process. An SLCDx 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 SLCDx. This is helpful in the case where one PC serial port is used for both download and serial control (e.g. HyperTerminal).

Quit

Exits BMPload.

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 alphabetized.

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 "Demo\Landscape" or “Demo\Portrait” 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 SLCDx" is clicked, it will be filled in with the CRC reported by the SLCDx 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 SLCDx 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 SLCDx serial ports can be set to 230400 baud and 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 SLCDx 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 SLCDx 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.

These are the limits on the macro commands and their arguments:

- MAXIMUM NUMBER OF MACROS = 254

  A macro can use an unlimited number of labels, which is an effective way to work around this limit.
- MAXIMUM MACRO NAME LENGTH = 64 chars
- MAXIMUM CALL DEPTH = 4

  A macro can call another macro, but only to a depth of 4.
- MAXIMUM ARGUMENTS PER MACRO = 10
- MAXIMUM BYTES PER ARGUMENT is only limited by the total command line length (max 128)
- MAXIMUM TOTAL STORED ARGUMENTS = 100

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 SLCDx 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 commands, and macro labels

Memory Commands

Memory commands were added to implement the keyboard in the demo macros that come installed with the SLCDx 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 to append a string to a string variable, and mpop` to remove characters from the end of string variable.

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

A special label, ":default", can be used to execute commands when the given label identifier is missing or does not match any other labels within the macro. This label must always be the last label in a macro.

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, to invoke label "attach" in macro number 8, the command would be:

m 8:attach

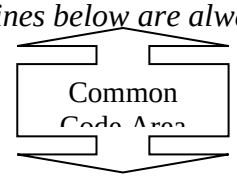
The format of a macro label invocation is: "m <macro name or number>:<label name>".

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

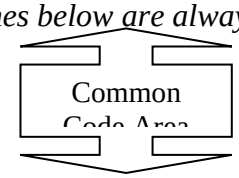
:define //*(execution stops here)*
t "Defining button" 200 50
bd 1 0 32 1 "INCREASE" 9 5 10 11

#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 SLCDx and run the command:

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

Now cycle power to the SLCDx, 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 command "`tf? 0`":

```
f 13B
S fff 000
ta LT
o 0 0
sc 0 0
tm T
t "this is a test" 120 130 X
y 300
f 13B
S 000 fff
ta LT
o 0 0
sc 0 0
tm T
t "this is a test" 120 130 X
y 300
```

6. FONTS

The SLCDx has built-in fonts (described in sections 6.2 and 6.3) and the ability to use external fonts (described in section 6.1). The BMPload program is used to download external fonts (including Unicode fonts) into Flash memory on the SLCDx board.

6.1 External Fonts

External fonts allow users to use fonts which are available in MS Windows. These programmable fonts are used often used to get a particular look-and-feel for an application, or to support a non-Latin language for product localization with. These fonts are loaded into the on-board flash memory along with bitmaps and macros. A font is contained in a .SIF extension file. A separate .SIF file is needed for each unique combination of font, size, and attribute (e.g. bold).

The basic steps to creating and using External fonts are:

1. Request a System Independent Font (.SIF) file from REACH Technical Support: Specify Windows font name, size (height in pixels or points), and attributes (bold, italic, both), code set ("ASCII + ISO 8859" (256 characters) or Unicode Character set), or individual characters*.
2. REACH will generate a .SIF from your specifications.
3. Create a Font List (.RFL) file: Create a text file with a ".RFL" file extension.
    a. The contents should contain a line for each font in the format: `<font_alias> <filename>.sif <CR>`
    b. There must be a space character between font alias and the filename.
    c. The limit for the font alias is 8 characters.
    d. The filename must include "aa2" or "aa4" (case insensitive) if the font uses 2bpp or 4bpp anti-aliasing.
4. Download Font List file: Use the Windows BMPload application under Windows.
5. Use the External font: Use the SET FONT command ("f <font_alias>") before any text related command.
6. If the character set is non-Latin Unicode, you will need to use the SET UTF8 ENCODING command before displaying any characters.

*Users who would like a .SIF file with individual characters (rather than a range of Unicode values) should submit a MS Notepad generated text file known as a “Pattern File”. To generate this file, copy desired characters into MS Notepad. When complete, use menu option “File->Save As”, with Encoding set to Unicode.

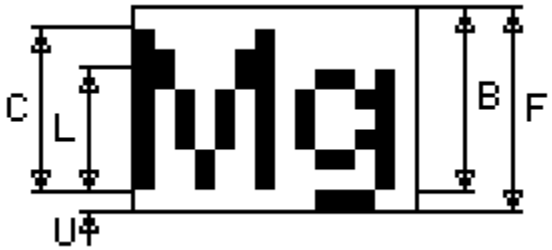
6.2 Proportional Fonts

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

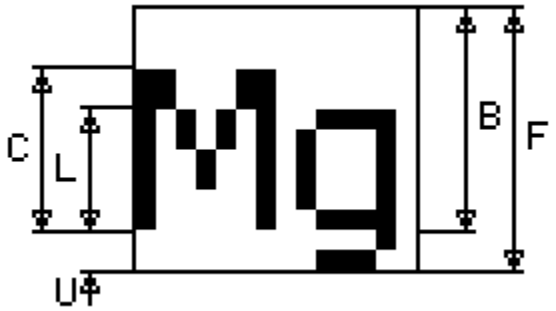


Font 10S – ISO 8859-1 (Latin1 or Western European)

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

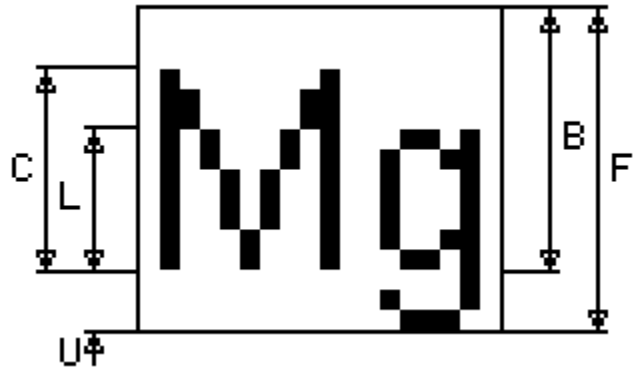


Font 13B – ISO 8859-1 (Latin1 or Western European)

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

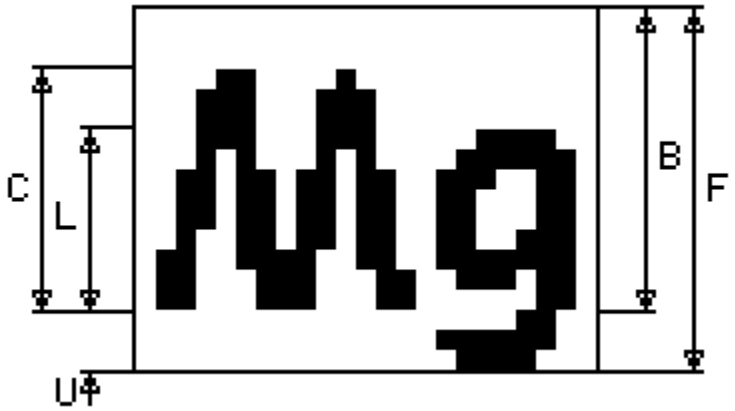


Font 16B – ISO 8859-1 (Latin1 or Western European)

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

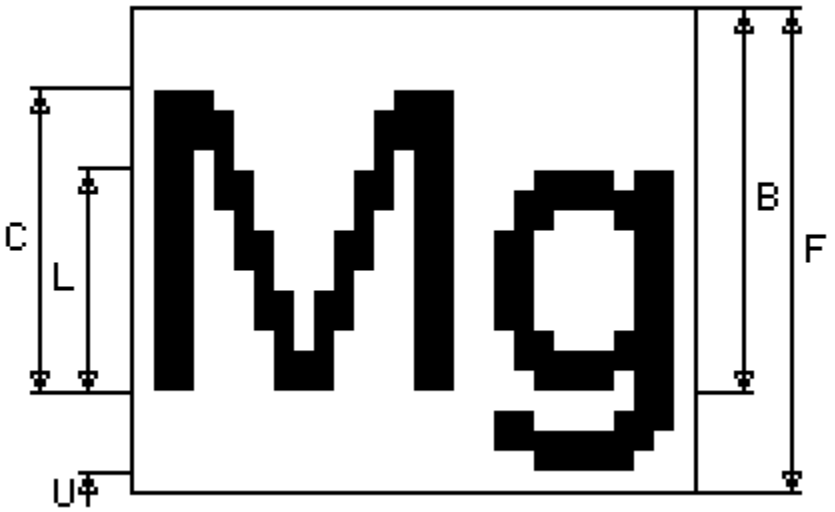
Font 18BC – ISO 8859-1 (Latin1 or Western European)

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



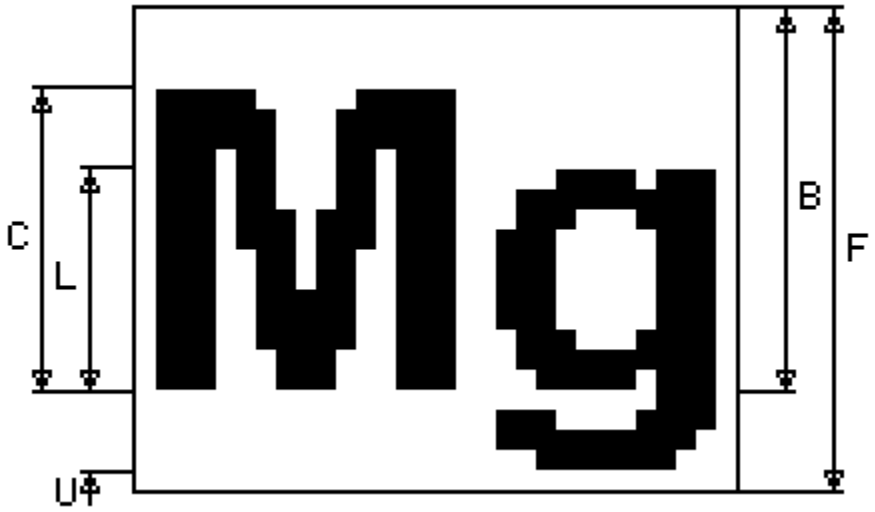
Font 24 – ISO 8859-1 (Latin1 or Western European)

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



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

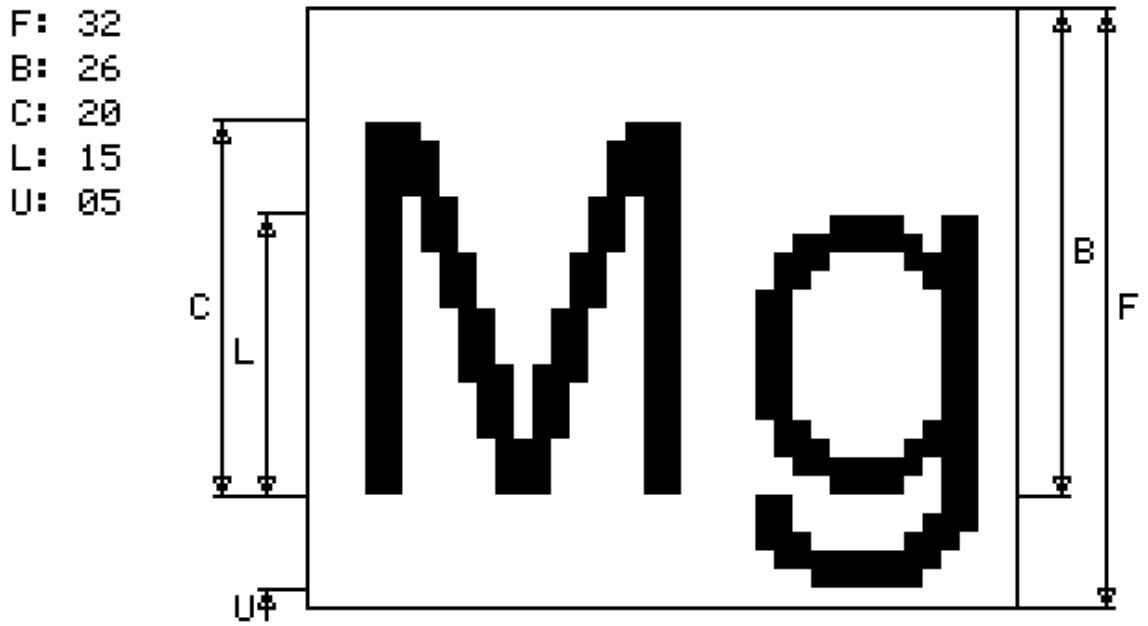
F: 24
B: 19
C: 15
L: 11
U: 04



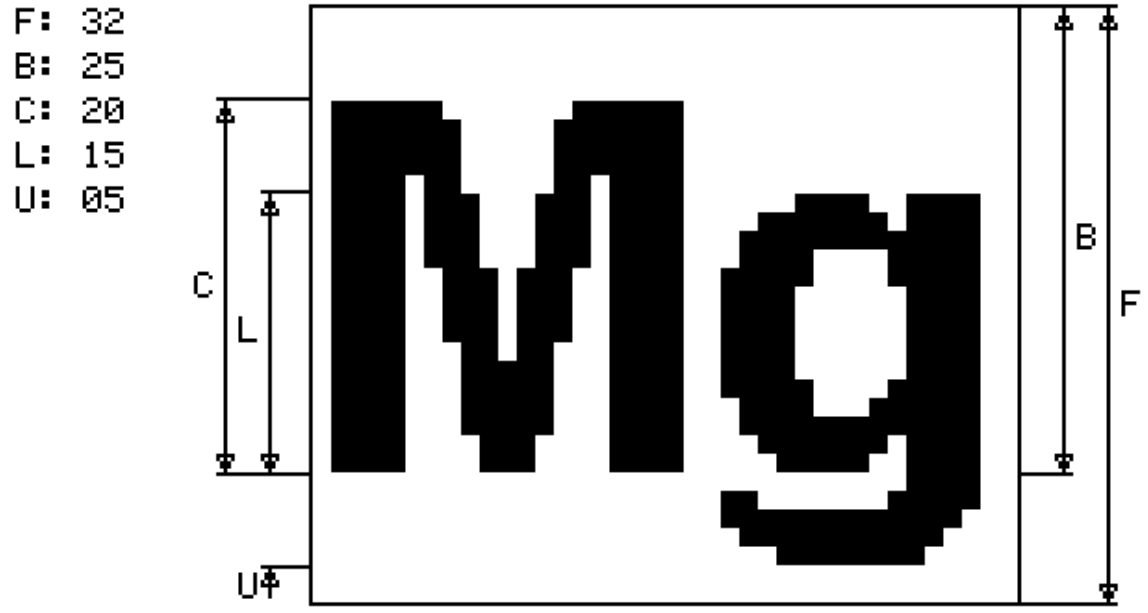
Font 24BC – ISO 8859-1 (Latin1 or Western European)

F: 24
B: 20
C: 17
L: 13
U: 04

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



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



6.3 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

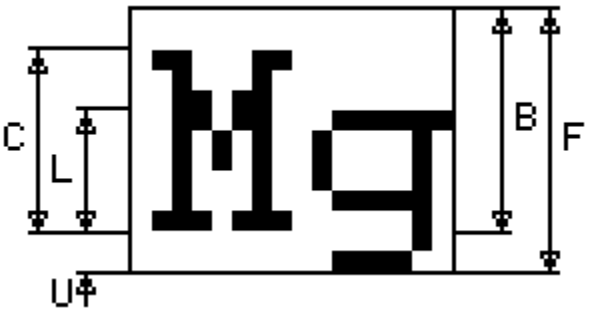
F: 10
B: 09
C: 09
L: 07
U: 01

Font 8x12 – ASCII Only

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

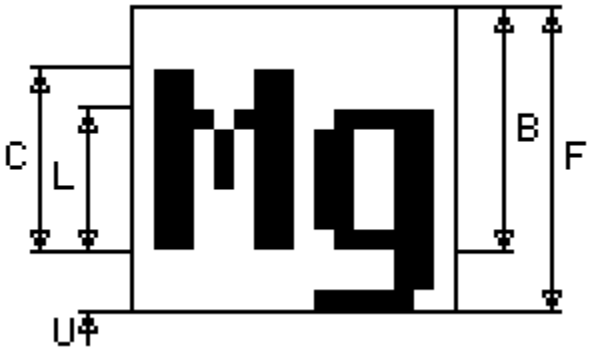
Font 8x13 – ASCII Only

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



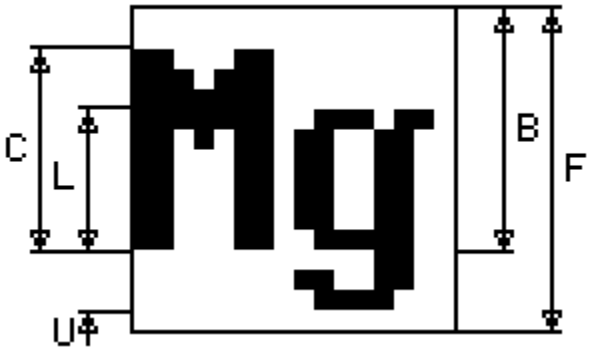
Font 8x15B – ASCII Only

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



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

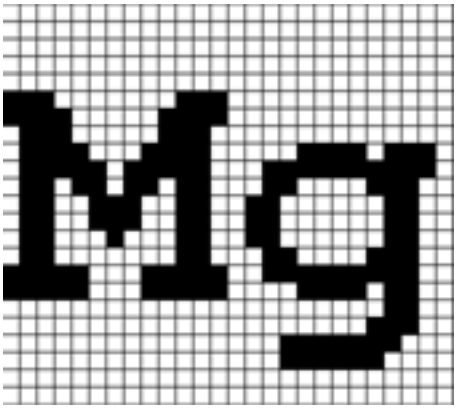
F: 16
B: 12
C: 10
L: 07
U: 03



Font 8x16L

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

Font 14x24 – ISO 8859-1



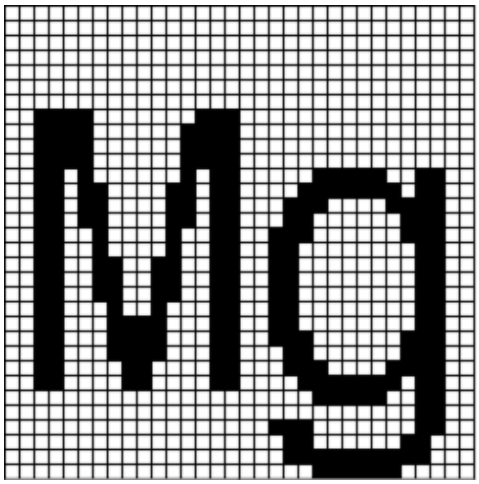
Font 16x32 – ISO 8859-1

This is the font 8x16 doubled in both directions:

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

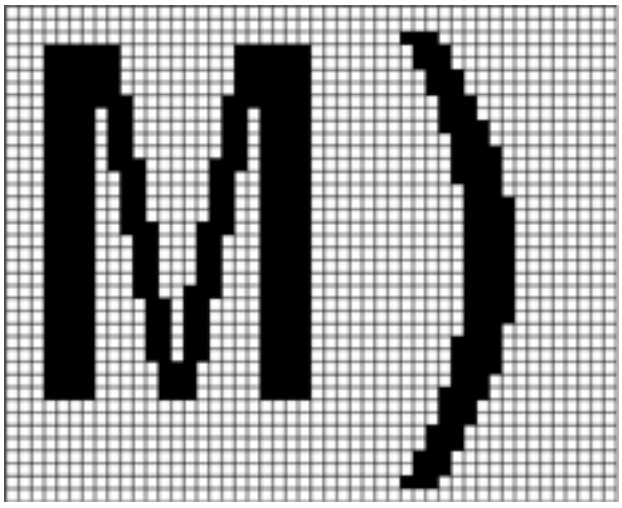
Font 16x32i – ISO 8859-1

This is an improved version of the 8x16 above.



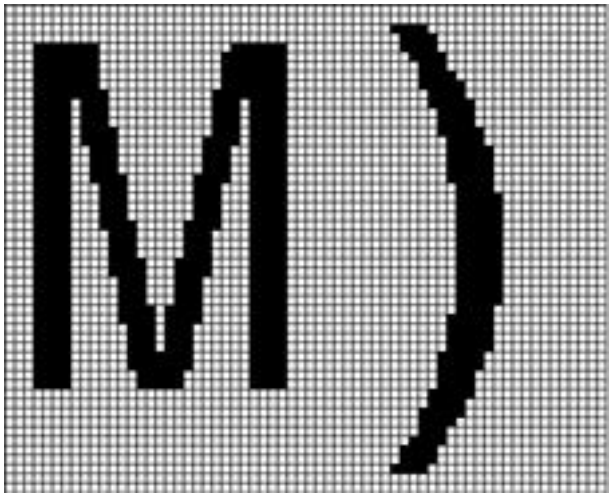
Font 24x48 – Numbers, Capital letters, Symbols

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



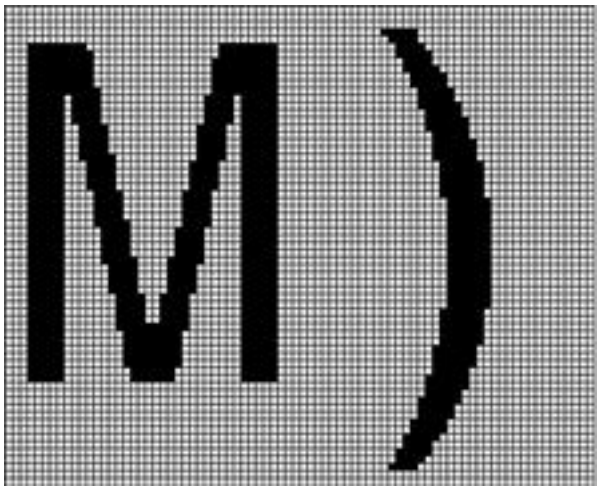
Font 32x64 – Numbers, Capital letters, Symbols

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



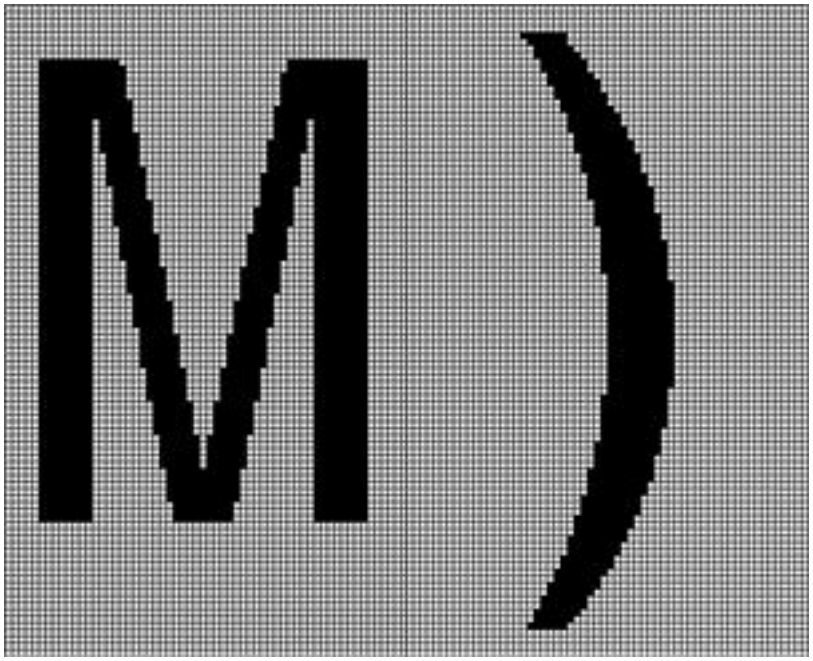
Font 40x80 – Numbers, Capital letters, Symbols

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.



6.4 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.

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
Đ	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

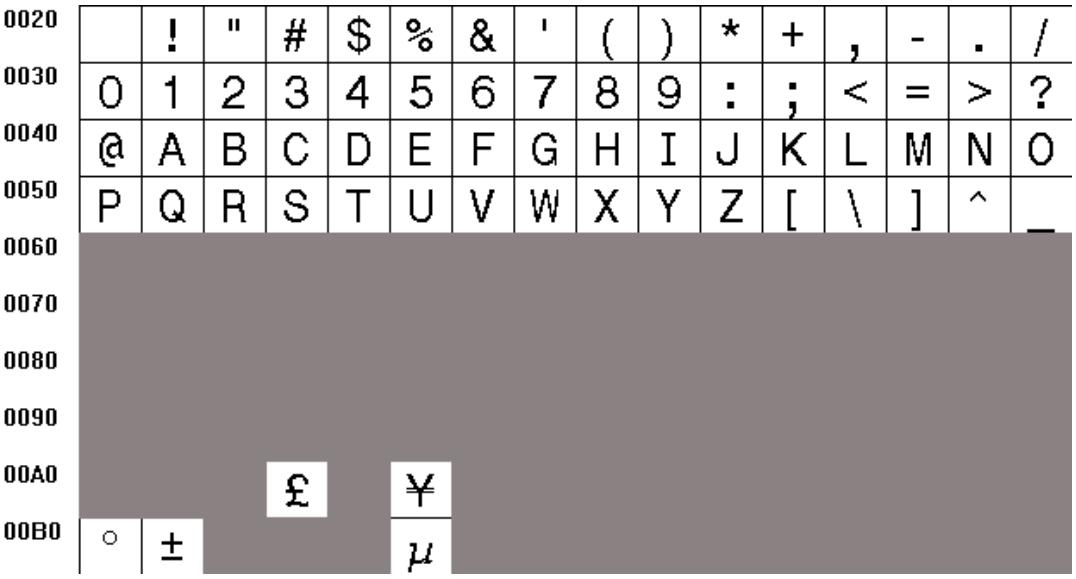
Char	Code	Name	Description
ð	240	eth	eth
ñ	241	ntilde	n tilde
ò	242	ograve	o grave
ó	243	oacute	o acute
ô	244	ocirc	o circumflex
õ	245	otilde	o tilde
ö	246	ouml	o umlaut
÷	247	divide	division sign
ø	248	oslash	o slash
ù	249	ugrave	u grave

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

6.5 Character Set - Numbers, Capital letters, Symbols

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



7. COLOR SPECIFICATIONS

7.1 Overview

The SLCD5 has several types of color specifications. Commands that use a color specification will refer to one of the specifications shown below.

Name	Color Depth	Description
RGB444	12-bit color	RGB, where each color is four bits, 0-F in value. Pure red would be F00, pure green would be 0F0 and pure blue would be 00F.
RGB565	16-bit color	RGB, where R is 5 bits, G is 6 bits, and B is 5 bits. Pure red would be F800, pure green would be 07E0 and pure blue would be 001F. See RGB565 Encoding.
RGB888	24-bit color	RGB, where each color is eight bits, 00-FF in value. Pure red would be FF0000, pure green would be 00FF00 and pure blue would be 0000FF.

7.2 RGB565 Encoding

The RGB565 color specification compresses Truecolor 24-bit colors into 16-bit color values. In this format, 5 bits of color are used for RED, 6 bits for GREEN, and 5 bits for BLUE.

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

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)

Bitwise, this is the same as:

4. Drop the three lowest bits of the Red and Blue values: 0b76543~~210~~
5. Drop the two lowest bits of the Green values: 0b765432~~10~~
6. Shift and OR the truncated R, G and B values into a 16-bit value in the form 0bRRRRRGGGGGGBBBBB

8. DEMO KIT TUTORIAL

8.1 *Connection and Control Via PC*

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

In order get acquainted with the SLCDx 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 SLCDx 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 SLCDx has been connected to an available serial port, open Hyperterminal (Programs->Accessories->Communications->Hyperterminal) and enter SLCDx for the name of the connection. Then enter the serial port connected to the SLCDx 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 SLCDx 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 SLCDx 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 SLCDx 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.

8.2 *Simple Commands*

This section presents some simple commands that illustrate some of the SLCDx 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
```

8.3 *Macros*

The SLCDx 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.

8.4 *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 WORKING WITH BITMAPS.
2. Create a 320x240 (SLCD+/6) pixel canvas (or 480x272 for SLCD43) 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 SLCDx command to locate the image and text.
3. Download the bitmaps using BMPload.
4. Write a series of SLCDx 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 SLCDx control programs. Visit our ***Download Center*** at http://www.reachtech.com. Sign up for an account and enjoy the wealth of technical information we have available for download.

8.5 *A Warning Concerning Commands That Affect Non-Volatile Settings*

Commands marked with a diamond symbol (**♦**) affect settings stored in non-volatile memory; this memory is implemented using an I2C Serial EEPROM chip with a limit of 1 million write cycles.

At 1 write per second, the part will be destroyed in 11 days. Take care that commands marked as affecting non-volatile settings are executed as seldom as possible.

9. WORKING WITH BITMAPS

9.1 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 SLCDx. The way to do this is to create or capture the visual element and create the desired layout.

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

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

NOTE: to make sure a bitmap image will fit on the SLCDx's screen, the Width and Height of the bitmap must each be less than or equal to the pixel resolution of the SLCDx:

- SLCD43 (WQVGA, 480x272)
- SLCD+, SLCD6 (QVGA, 320x240)

9.2 8 bit Color Mode - SLCD+

The SLCD+ 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.

9.3 Highcolor Mode - SLCD6, SLCD43, SLCD+

All SLCDx controllers support Highcolor mode as standard. These can 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. See RGB565 Encoding for details.

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

Note: The SLCD+ can support 8 bit color if needed for backward compatibility with old (legacy) SLCD controllers.

9.4 Transparency (Highcolor Mode Only)

In Highcolor Mode transparent bitmaps are supported. To make a bitmap transparent, select a transparency color. It must have an exact RGB565 equivalent, e.g. solid red 0xF800 (top 5 bits), solid green 0x07E0 (middle 6 bits), or solid blue 0x001F (lower 5 bits). Then have the bitmap file name include the string ".unc.trXXXX ", where XXXX is the 16 bit RGB transparent color value. For example, if solid red is transparent, have the filename include the form above, such as "01_my_bitmap.unc.trf800.bmp". Note, the lower-case ".unc" indicates that the bitmap is uncompressed, which is required for transparency. When this bitmap is displayed, the transparent color will not be displayed.

10. IN-SYSTEM BITMAP AND FONT DOWNLOAD

10.1 Introduction

Starting with version 2.5 of the SLCDx 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 SLCDx 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 SLCDx will not respond to normal commands.

Once the bdld command has been issued, the SLCDx 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 SLCDx 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 SLCDx 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 SLCDx responds as expected. If, however, the CRC check fails, the SLCDx ignores the command and returns a invalid CRC response ('#'<return>).

11.2 Command Protocol

The format for a CRC'd command is:

```
~<CRC><SLCDx Command><return>
```

A '~' (tilde) character at the start of the command string signals the SLCDx that an embedded CRC (4 ASCII-Hex chars, [0-9,a-f,A-F]) will follow the '~' and then the actual SLCDx command will begin. The CRC is calculated for the SLCDx 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 SLCDx 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 SLCDx 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 SLCDx. 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 "SLCDx Command Line<CR>"
// output: "~HHHHSLCDx 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. COMMUNICATIONS WATCHDOG TIMER

12.1 Overview

The SLCDx firmware contains a Communications Watchdog Timer feature that can be used to execute a macro with one of four predefined labels when a corresponding communications event is detected:

7. Short Term loss of communication
   - No characters received for X seconds
   - Specified macro executes with label ":st_down"

8. Communication restored after Short Term loss
   - A character is received after a Short Term loss occurred
   - Specified macro executes with label ":st_up"

9. Long Term loss of communication
   - No characters received for Y seconds
   - Specified macro executes with label ":lt_down"

10. Communication restored after Long Term loss
    - A character is received after a Long Term loss occurred
    - Specified macro executes with label ":lt_up"

The macro to execute and the definitions of Short Term and Long Term communications loss are specified using the '*comwdt' command, which also enables the feature. Once enabled, it remains active until a '*comwdt off' command is received, power is cycled or the SLCDx is reset.

*12.2 Using the '*comwdt' Command*

Command: `*comwdt <macro> <short> <long>`

Arguments:

`<macro>`	–	name or index of macro to execute
`<short>`	–	Short Term timeout, in seconds
`<long>`	–	Long Term timeout, in seconds

12.3 Example

Below are example macro definitions demonstrating a simple usage of the Communications Watchdog:

```
// example macro to enable the watchdog
// short term timeout is 5 seconds, long term is 10 seconds
//
#define enable_wdt
z
f32B
ta CC
t "Communications Timeout Demo\n" 240 25
*comwdt com_wdt 5 10
#end
// macro that gets executed when comwdt times out
// predefined labels are used for the timeout actions:
//
#define com_wdt
//
// the following is a required label, do not change the name
:st_down
// short term timeout expired; animate backlight blinking
ani 0 xbbs 127
ani 0 y 250
ani 0 xbbs 255
ani 0 y 250
anie 0
//
// the following is a required label, do not change the name
:st_up
// recover from short term timeout; kill the animation and
// restore to full brightness
anix 0
xbb 255
//
// the following is a required label, do not change the name
:lt_down
// long term timeout expired; clear screen and warn user
anix 0
xbb 255
z
f32B
ta CC
t "Communications Timeout\n" 240 75
ta CC
t "Please Restart System"
//
// the following is a required label, do not change the name
:lt_up
// recover from long term timeout; in a real system the power-on macro or some other startup
// macro would be run
z
f32B
ta CC
t "Communications Restored" 240 200
#end
// example macro to disable the watchdog
```

```
//
#define disable_wdt
z
f32B
ta CC
t "Communications Timeout Disabled\n" 240 20
*comwdt off
#end
```

13. WORKING WITH VARIABLES

13.1 Overview

As of version 2.6.13, the SLCDx has supported a limited set of variables with primitive access capabilities. As of version 2.7.20, simple printf() like formatting is supported; this allows using variables in formatted output on the SLCDx display or in a line of text sent to one of the serial ports. As of version 2.9.38, the number of integer variables (i0-i9) has been increased to a total of 20 (i0-i19).

13.2 User Variables

User Variables are available with predefined names which specify their data type, as shown in the table below. They are assigned values using the "Set Variable" command, and the host can query their values using the "Get Variable" command. Their values can also be used as arguments to commands or macros by enclosing their names inside back-tick characters.

NOTE: the back-tick is a different character than the single-quote/apostrophe character.

Variable Name	Format Type	Data Type
i0 – i19	numeric	32-Bit integer
e0 – e9	numeric	8-bit unsigned integer (see the EEPROM warning)
s0 – s9	string	Text string, max of 80 chars
p0 – p9	numeric	Pair of X and Y Coordinates (16-Bit integers)

13.3 System Variables

System Variables have values that are set by the system and *cannot* be accessed via the "get" and "set" commands. Their values *can* be used as arguments to commands or macros by enclosing their names inside back-tick characters.

Variable Name	Format Type	Data Type
H#	numeric	Height of bitmap # (in pixels)
W#	numeric	Width of bitmap # (in pixels)
V	numeric	Volume setting (0--255)
B	numeric	Brightness setting (0--255)
M	string	Default "mpush" string, max of 80 chars
T	string	5 char string containing SLCDx's temp in degrees C; formatted "%+05.1d", the last 2 chars will always be the decimal point and the tenths digit.
L128 -- L255	numeric	Slider value (16-Bit integer)
R#:#	numeric	Random integer in the range #:#
Xa, Ya, Xr, Yr	numeric	Last Touch Coordinates, absolute and relative to hotspot Top Left Corner (16-Bit integers)
Xc, Yc, Xm, Ym, Xs, Ys, Xo, Yo	numeric	Absolute X and Y coordinates for center of display, bottom right corner of display, screen cursor as set by the 'sc' command, and origin.
$BAUD0	numeric	String containing Main Com Port Baud Rate

13.4 Formatting Variables

A simple printf()-like output format may be applied to a variable by inserting a format specifier between the 1st back-tick and the variable's name, with a space between them, as shown in the examples below. Two different format specifier syntaxes are supported; one for string variables, and one for numeric variables (see Format Types in preceding tables). NOTE: macro arguments '0'--'9' must be formatted using the string syntax.

- string syntax: `` `%[-]<width>[.<max>] <varName>` ``

  `[-]` Left justify output text by adding trailing spaces; default is right justification by adding leading spaces

  `<width>` Minimum output field width (1-79)

  `<max>` Max. number of characters from string variable to output (1-79)

  Examples:

  ```
  set s0 "abcdef"
  ```
  Sets string variable s0 to "abcdef"

  ```
  get s0
  abcdef
  ```
  Outputs string variable s0

  ```
  out "[`%-4.3 s0`]\r"
  [abc ]
  ```
  Outputs "[", up to 3 chars from s0 in a left justified field of 4 chars, "]", and a return.

  ```
  out "[`%4.3 s0`]\r"
  [ abc]
  ```
  Outputs "[", up to 3 chars from s0 in a right justified field of 4 chars, "]", and a return.

  ```
  out "[`%6 T`]\r"
  [ +33.1]
  ```
  Outputs "[", the temperature string in a right justified field of 6 chars, "]", and a return.

  ```
  out "[`%6.3 T`]\r"
  [   +33]
  ```
  Outputs "[", 3 chars from the temperature string in a right justified field of 6 chars, "]", and a return.

- numeric syntax: `` `%[+|-|0|+0]<width> <varName>` ``

  `[+]` Always include sign of value.

  `[-]` Left justify output text by adding trailing spaces; default is right justification by adding leading spaces.

  `[0]` Right justify by adding leading 0's.

  `[+0]` Always include sign of value AND right justify output by adding leading 0's after the sign.

  `<width>` Minimum output field width

  Examples:

```
set i0 123
```

Sets integer variable i0 to positive 123

```
get i0
123
```

Outputs integer variable i0

```
out "[`%+06 i0`]\r"
[+00123]
```

Outputs "[", the value of i0 in a right justified field of 6 chars with the sign of i0 followed by enough leading 0's to fill out the field , "]", and a return.

```
out "[`%+6 i0`]\r"
[  +123]
```

Outputs "[", the value of i0 in a right justified field of 6 chars with the sign of i0 preceded by enough leading spaces to fill out the field , "]", and a return.

```
out "[`%-6 i0`]\r"
[123   ]
```

Outputs "[", the value of i0 in a left justified field of 6 chars with enough trailing spaces to fill out the field , "]", and a return.

```
set p0 12 345
out "[`%06 p0`]\r"
[000012 000345]
```

Outputs "[", the X and Y values of p0 in right justified fields of 6 chars with enough leading 0's to fill out the fields and a space between fields, "]", and a return.

14. USING SIMPLE MATH EXPRESSIONS

14.1 Overview

Since version 2.7.0, the SLCDx has supported a limited simple math capability, using the backtick-escaped format:

`(`<argNum>`<op><value>)`, where:

- <argNum> can be '0'-'9', representing macro arg `0`-`9`
- <op> can be '+' or '-' for addition, or subtraction
- <value> can be 0-65536

Such expressions could be used only within a macro, and were thus of limited value.
Now, in version 2.9.0 and later, simple math expressions can be used with any command that accepts integer values as arguments**.** When the command executes, the expression will be evaluated and its numeric value will be used. The new format is:

`(<value><op><value>)`, where:

- <value> can be:
  - A positive number between 0 and 65535
  - A backtick-escaped macro argument (`0`-`9`) or the backtick-escaped name of any variable with a numeric format type, as listed in WORKING WITH VARIABLES (above).
- <op> can be one of:
  - '+' for addition
  - '-' for subtraction
  - '*' for multiplication
  - '/' for division (integer, truncating)
  - '%' for integer modulo

14.2 Limitations and requirements

- No nesting of expressions is allowed (to avoid problems with recursion).
- BMPload for SLCD+/SLCD6/SLCD43, version 1.11.0 or later is required.
- Expressions used in an "ani" command or a "tf" command will be evaluated before the resulting command is stored in the animation buffer.

14.3 Examples

Below are some example macro definitions demonstrating valid variants of the new format:

```
// Display a given bitmap centered on the screen:
// -- arg 0 is index of bitmap
// -- arg 1 is its width
// -- arg 2 is its height
//
#define showBitmapCentered
set i0 `(`1`/2)`      // divide width by 2
set i0 `(`Xc`-`i0`)`  // subtract from Screen Center X coord
set i1 `(`2`/2)`      // divide height by 2
set i1 `(`Yc`-`i1`)`  // subtract from Screen Center Y coord
xi `0` `i0` `i1`      // display the bitmap
#end

// show bitmap 1 in center of screen
//
#define test_sbc
m showBitmapCentered 1 `W1` `H1`
#end

// increment variable i0 each time called:
//
#define inc_i0
set i0 `(`i0`+1)`
#end
```

15. 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.
6/5/2009	Added information about antialiasied fonts.
7/22/2009	Clarified Notes in METER DEFINE.
8/03/2009	Added Chart Type 3 (STRIP starting at RIGHT EDGE) and clarified CHART DEFINE BITMAP example.
8/12/2009	Changed to new corporate address.
8/18/2009	Added information on Transparent Bitmaps.
8/25/2009	Added WAIT FOR REFRESH command. Corrected terms in WAIT VERTICAL RETRACE command.
9/8/2009	Updated command name and content: CLEAR HOTSPOT. Changed name: CLEAR ALL HOTSPOT.
9/17/2009	Added "Working with Variables" section; added commands "xst", "xxy", "xim"; removed Circular Slider; added x parameter to "xset"; clarified Draw Rectangle usage.
10/12/2009	Added "text in a rectangle" form of the "t " command; added subsection to describe pixel resolutions of the supported LCD panels.
01/11/2010	Added subsection describing use of External Fonts; added "*touchParm" cmd; added explanation of predefined labels for TOUCH MACRO ASSIGN commands "xm" and "xmq"; added brief mention of commands "xma", "xmc", and "xme".
01/14/2010	Added optional [index] parameter to "lsmac" and "lsbmp" commands.
02/02/2010	Made [Yield #] optional for "anid" command.
02/05/2010	Corrected grammar, clarified Note for "anid" command; cleaned pagination.
06/21/2010	Extended description of "cv" command and examples; did same for "t" command.
04/18/2011	Added Chart commands ‘cc’, ‘cr’, ‘cdp’, ‘cdbc’, ‘cdbb’, ‘cdrm’, ‘cddm’
02/03/2012	Removed Section on RS485 commands - deprecated
03/16/2012	Added section for Communications Watchdog Timer, hyperlinks to new section, fixed typos in various places, fixed various broken hyperlinks, fixed formatting in several places to match existing formatting. Removed comments about the ‘beep’ from xaq and xmq commands. Added documentation for *touchMode command.
06/04/2012	Removed index=5 support in bdld command, fixed description of cdbb command, added *speedtest command, formatting cleanup. Added bbmp command description.
07/30/2012	Sorted commands by name, reconciled command names with SLCD5 Manual, added tables of command names with cross-references, cleaned up format throughout the manual and updated command descriptions where needed.
01/21/2013	Added commands for Scrolling Textbox feature.
02/20/2013	Fixed example for “xim” command.
05/20/2013	Clarified that the Scrolling Textbox commands are for the SLCD43 only. Added I2C Read, I2C Write, and I2C Speed commands. Clarified that uncompressed bitmaps have a lower-case “.unc” in the file name. Fixed folder referenced in Bitmap Order description.
05/22/2013	Fixed typo in PREPEND SCROLLING TEXTBOX description.
05/12/2013	Removed ‘external’ from font description from TEXT DISPLAY command wrap/rotate note, added clarification about the xset ‘T’ option, clarified arc segment parameters, corrected value limit in Simple Math Expressions.
06/14/2013	Fixed slider index range specification, added mdb and mvb commands.

07/23/2013	Added callout to EEPROM warning for “e” variables.
07/26/2013	Added many missing hyperlinks for unlinked references, added Color Specifications section, changed wording of all ‘color’ parameters to use the new color specifications.
07/29/2013	Fixed typos in the RGB888 color description, clarified IC location for READ TEMPERATURE command.
08/01/2013	Added xsnb, xxynb commands, increased range of ‘I’ variables to i19, added ssb query.
08/09/2013	Fixed connector reference for xbl command.
10/22/2013	Added getx and setx commands. Added range of <count> value in “beep” commands.
10/25/2013	Added EEPROM warning to *eew command. Added Reply detail to the *touchMode command.
10/30/2013	Added “all” and range options to xd/xe commands.
11/05/2013	Clarified/fixed descriptions for cdbc, cddm, and cdrm commands.
04/10/2014	Added clarification about vertical alignment in SET TEXT ALIGN command, added information about text alignment in TEXT DISPLAY command.
04/14/2014	Clarified initial state of type 2 latching state button.
04/16/2014	Added reference to xset command for hotspot and button characteristics, added xset command example. Fixed xset command listing syntax.
04/22/2014	Added note that XOR mode (for drawing text) does not work with anti-aliased fonts.
04/24/2014	Added clarifications on parameter types in METER DEFINE, CHART DEFINE, LEVELBAR DEFINE, and SLIDER DEFINE commands.
05/14/2014	Added clarification to several drawing commands about using the current foreground color. Fixed some typos and text formatting. Added argument format clarification to SET COLOR (Detailed).
06/09/2014	Fixed minor typo in WAIT VERTICAL RETRACE description.
07/11/2014	Added clarification about bitmap numbering, added detail about error replies.
09/05/2014	Fixed typos in Formatting Variables section, fixed typo in ‘cr’ Command by Function listing.
09/17/2014	Clarified CHART REDEFINE PEN description, added missing commands to reference lists, fixed font/capitalization typos.
09/18/2014	Added command entries for mpush, mpop.
09/23/2014	Added *com? command, clarified text in description for Labels.
10/27/2014	Clarified value per pen requirement in CHART VALUES command.
11/20/2014	Added ‘get’ feature of PEN WIDTH command.
04/29/2015	Preparation for 2.10.0 firmware release. Added SET STATE BUTTON DELIMITER command, added SET I/O PIN (SLCD43 ONLY) command, added ALLOW PRE_EXISTING TOUCH commands.
08/27/2015	Enabled bookmarks, formatting changes to correct bogus bookmarks.
09/21/2015	Fixed hyperlinks for *apt and *aptnb commands in xref.
12/18/2015	Corrected LED called out in the SET LED command description.
02/23/2016	Corrected descriptions for GET VARIABLE and GET VARIABLE (HEX) to clarify integer variables up to i19.
05/12/2016	Added clarification to xset command description. Corrected font reference for DEFINE SCROLLING TEXTBOX. Clarified state definitions of latching state buttons.

05/13/2016	Fixed various navigation links in index tables.
06/01/2016	Changed company name.
01/17/2017	Fixed minor formatting errors in BUTTON DEFINE CENTER TEXT command description.
04/14/2017	Changed Sales phone number and company mailing address to San Jose location.
06/15/2020	Changed copyright information.
03/25/2021	Changed sales phone number.
07/27/2022	Changed RMA address & company address.