
SLCD5+N Reference Manual

Version 1.3

42-0323-xxx
Variations 001/101/301

Firmware Version 1.5.0 and above
BMPload 2.4.0 and above

02/29/2016
Hardware Revision 02

Note: This manual refers to the SLCD5+N controller board. Specifications for various Display Module products that use this controller board are found in the individual product datasheets. Please refer to your Display Module's datasheet for power requirements, input voltage, temperature ranges, etc.

© Copyright Reach Technology Inc. 2016
All Rights Reserved

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

Reach Technology Inc.

www.reachtech.com

Sales: 510-770-1417 x112
sales@reachtech.com

Technical Support: 503-675-6464
techsupport@reachtech.com

Table of Contents

0.	HARDWARE LIMITED WARRANTY AND SOFTWARE LICENSE AGREEMENT	10
0.1	HARDWARE LIMITED WARRANTY	10
0.2	RETURNS AND REPAIR POLICY	10
0.3	SOFTWARE LICENSE AGREEMENT	11
1.	INTRODUCTION	14
1.1	OVERVIEW	14
1.2	CONTROLLER FEATURES.....	14
1.3	BOARD PICTURES.....	15
	<i>42-0323-001, 42-0323-101</i>	<i>15</i>
	<i>42-0323-301.....</i>	<i>16</i>
1.4	ORDERING INFORMATION	17
1.5	ELECTRICAL CHARACTERISTICS	17
	<i>Backlight</i>	<i>17</i>
1.6	PANEL SUPPORT.....	17
1.7	SD CARD SUPPORT	18
1.8	DIMENSIONS.....	18
2.	CONNECTORS AND JUMPERS.....	19
2.1	OVERVIEW	20
2.2	J6 – POWER AND COMMUNICATION COM0 (RS232 MODE)	21
	<i>COM0 Cable.....</i>	<i>21</i>
2.3	J6 – POWER AND COMMUNICATION COM0 (3.3V CMOS MODE)	22
2.4	J7 – COMMUNICATION COM1 AND RESET	23
	<i>COM1 Cable.....</i>	<i>24</i>
2.5	J4 – POWER	25
2.6	J9 – 33 PIN 0.5MM FLAT FLEX (HARDWARE VERSION DEPENDENT)	26
2.7	J2 – LVDS LCD PANEL (HARDWARE VERSION DEPENDENT)	27
2.8	J3 – 4 WIRE TOUCH.....	28
2.9	J1 – 4 WIRE TOUCH.....	28
2.10	J10, J17 – PCAP I2C TOUCH SENSOR (HARDWARE VERSION DEPENDENT).....	28
	J13 – BACKLIGHT / INVERTER CONTROL (HARDWARE VERSION DEPENDENT).....	29
2.11	J11, J12 – ON-BOARD BACKLIGHT DRIVER (HARDWARE VERSION DEPENDENT)	29
2.12	J16 – JTAG	29
2.13	J5 – EXTERNAL AUDIO OUTPUT (HARDWARE VERSION DEPENDENT)	29
2.14	J15 SD CARD CONNECTOR	30
3.	CONFIGURATION GUIDE	31
3.1	POWER CONNECTIONS	31
3.2	SERIAL PORTS	32
3.3	TFT HARDWARE PANEL ORIENTATION.....	32
3.4	SYSTEM CONFIGURATION	33
4.	SYSTEM OVERVIEW	36
4.1	GENERAL SLCD5+N CONTROLLER INFORMATION.....	36
4.2	COMPATIBILITY WITH SLCD, SLCD6, SLCD43, SLCD+ CONTROLLERS.....	36

4.3	BITMAPS AND MACROS.....	36
4.4	OVERVIEW – SLCD5+N DEVELOPMENT KITS	37
4.5	GETTING STARTED.....	37
4.6	CONNECTING THE DEVELOPMENT KIT TO A PC	38
4.7	SWITCHING BETWEEN PC AND EMBEDDED CONTROLLER.....	38
4.8	DEVELOPMENT KIT OPERATIONAL NOTES	39
4.9	COMMUNICATIONS INTERFACE.....	40
4.10	INPUT BUFFER PROCESSING.....	40
	Input Buffer	40
	Flow Control	41
	Buffer Limit Discussion	41
	Prompt Summary	42
4.11	TOUCH INTERFACE.....	42
	Hotspot	42
	Button	42
	Host Notification	43
4.12	HOST INPUT PROCESSING	43
4.13	SYSTEM FIRMWARE BOOT PROCESS	43
	Algorithm	44
5.	SOFTWARE COMMAND REFERENCE.....	46
5.1	COMMANDS AFFECTING NON-VOLATILE SETTINGS (EEPROM).....	46
5.2	COMMAND BASICS.....	46
	Compressed Syntax	46
	Case Sensitivity	46
	Assumed Orientation	47
	Touch Priority	47
5.3	COMMANDS BY FUNCTION.....	47
5.4	COMMANDS ALPHABETICALLY BY COMMAND SYNTAX.....	54
5.5	COMMANDS.....	60
	ALARM	60
	ANIMATED GIF START	60
	ANIMATED GIF STOP	60
	ANIMATION CLEAR (TEXT FLASHING CLEAR)	61
	ANIMATION DEFINE	61
	ANIMATION DELETE	63
	ANIMATION DISABLE	64
	ANIMATION ENABLE	64
	ANIMATION LIST (TEXT FLASHING LIST)	64
	ANIMATION SYNCH	65
	ANIMATION YIELD	65
	APPEND TO SCROLLING TEXTBOX	66
	BEEP FREQUENCY, BEEP FREQUENCY SPECIAL	66
	BEEP ONCE	66
	BEEP REPEAT	67
	BEEP TOUCH	67
	BEEP VOLUME, BEEP VOLUME SPECIAL	67
	BEEP WAIT	68
	BINARY CHART VALUES	68
	BINARY DOWNLOAD	69
	BINARY NOTIFICATION MODE	70

BUTTON CLEAR	70
BUTTON DEFINE CENTER TEXT	71
BUTTON DEFINE – LATCHING STATE	72
BUTTON DEFINE – MOMENTARY	73
CHANGE SD CARD DIRECTORY	74
CHART CLEAR DISPLAY AREA	74
CHART DEFINE	75
CHART DEFINE with BITMAP	76
CHART REDEFINE BACKGROUND BITMAP	76
CHART REDEFINE BACKGROUND COLOR	77
CHART REDEFINE CLEAR-BEFORE-DRAW MODE	77
CHART REDEFINE PEN	77
CHART REDEFINE RETRACE MODE	77
CHART RESET PENS TO START	77
CHART VALUES	78
CLEAR ALL HOTSPOT	78
CLEAR HOTSPOT	78
CLEAR SCREEN	79
CLEAR SCROLLING TEXTBOX	79
CLEAR SCREEN SPECIAL	79
CLEAR TEXT WINDOW	79
CONTROL PORT AUTOSWITCH	79
CONTROLLER TYPE	80
COPY FLASH to DRAM	80
COPY TEXT WINDOW VISIBLE TO VARIABLES	80
CRC EXTERNAL FLASH	81
CRC PROCESSOR BOOTLOAD CODE	81
CRC PROCESSOR CODE	81
CRC SCREEN	81
DEBUG MACRO	81
DEBUG TOUCH	82
DEFINE HOTSPOT (VISIBLE TOUCH AREA)	82
DEFINE INVERTER CONTROL POLARITY	82
DEFINE INVERTER PWM CONTROL	83
DEFINE MAX, MIN BRIGHTNESS	83
DEFINE PANEL ORIENTATION	84
DEFINE RELATIVE X-Y HOTSPOT	84
DEFINE RELATIVE X-Y HOTSPOT (Silent – No Beep)	84
DEFINE SCROLLING TEXTBOX	85
DEFINE SPECIAL HOTSPOT (Invisible Touch Area)	85
DEFINE SPECIAL HOTSPOT (Invisible, No Beep)	85
DEFINE SPECIAL TYPOMATIC TOUCH AREA	85
DEFINE TEXT WINDOW	86
DEFINE TOUCH ACTION	86
DEFINE TOUCH CALIBRATION TIMEOUT	87
DEFINE TOUCH PARAMETERS	87
DEFINE TOUCH SIGNAL ORIENTATION	87
DEFINE TYPOMATIC TOUCH AREA	87
DISABLE TOUCH (Hotspot / Button)	88
DISPLAY BITMAP IMAGE	88
DISPLAY BITMAP IMAGE CENTERED	88

<i>DISPLAY CLIPPED BITMAP IMAGE</i>	89
<i>DISPLAY CONFIG STRING</i>	89
<i>DISPLAY IMAGE FILE</i>	90
<i>DISPLAY ON/OFF</i>	90
<i>DISPLAY WINDOWED BITMAP IMAGE</i>	91
<i>DRAW ARC SEGMENT</i>	91
<i>DRAW CIRCLE</i>	92
<i>DRAW ELLIPSE</i>	92
<i>DRAW FILLED ELLIPSE</i>	92
<i>DRAW FILLED POLYGON</i>	93
<i>DRAW LINE</i>	93
<i>DRAW OUTLINE POLYGON</i>	93
<i>DRAW POLYLINE</i>	93
<i>DRAW RECTANGLE</i>	94
<i>DRAW ROTATED FILLED POLYGON</i>	94
<i>DRAW ROTATED POLYGON</i>	95
<i>DRAW ROTATED POLYLINE</i>	95
<i>DRAW TRIANGLE</i>	95
<i>EEPROM READ, WRITE</i>	96
<i>ENABLE TOUCH (Hotspot / Button)</i>	96
<i>EXTERNAL BACKLIGHT BRIGHTNESS CONTROL</i>	97
<i>EXTERNAL BACKLIGHT ON/OFF</i>	97
<i>EXTERNAL MEMORY AVAILABLE</i>	97
<i>EXTERNAL MEMORY CHIP ERASE</i>	97
<i>GET PANEL TYPE</i>	97
<i>GET TEXT DISPLAY WIDTH IN PIXELS</i>	98
<i>GET VARIABLE</i>	98
<i>GET VARIABLE (HEX)</i>	99
<i>GOTO POSITION IN TEXT WINDOW (Absolute)</i>	99
<i>GOTO POSITION IN TEXT WINDOW (Percentage)</i>	100
<i>LAST FIRMWARE FILE LOADED</i>	100
<i>LEVELBAR DEFINE</i>	101
<i>LEVELBAR VALUE</i>	101
<i>LIST BITMAPS DETAIL</i>	102
<i>LIST DOWNLOADED RECORDS</i>	102
<i>LIST MACROS DETAIL</i>	102
<i>LIST SD CARD DIRECTORY</i>	102
<i>LOAD SYSTEM FILE FROM SD CARD (CURRENT DIRECTORY)</i>	103
<i>LOAD SYSTEM FILE FROM SD CARD (SPECIFIED DIRECTORY)</i>	103
<i>LOAD SYSTEM FILE FROM SD CARD INTO FLASH</i>	103
<i>LOAD TEXT WINDOW BUFFER</i>	104
<i>MACRO ABORT</i>	104
<i>MACRO EXECUTE</i>	105
<i>MACRO NOTIFY</i>	106
<i>MEMORY POP</i>	106
<i>MEMORY PUSH</i>	107
<i>METER DEFINE</i>	108
<i>METER DEFINE BAND</i>	109
<i>METER VALUE</i>	110
<i>METER VALUE BAND</i>	111
<i>MOVE TEXT WINDOW</i>	112

OUTPUT STRING (AUX)	113
OUTPUT STRING (MAIN)	113
PANEL TIMING ADJUST	113
PIXEL READ	114
PIXEL WRITE	114
POWER-ON MACRO	115
PREPEND SCROLLING TEXTBOX	115
QUERY SCROLLING TEXTBOX	116
QUERY TEXT WINDOW	116
READ FRAME BUFFER LINE	116
READ FROM AUX PORT	116
READ TEMPERATURE	117
REDRAW ROTATED POLYGON	117
REDRAW TEXT WINDOW	117
RESET BOARD / SOFTWARE	117
RESET BOARD TO MANUFACTURED STATE	118
RESET TOUCH CALIBRATION	118
RESTORE DRAWING ENVIRONMENT (State Restore)	118
SAVE DRAWING ENVIRONMENT (State Save)	119
SAVE CONFIGURATION TO SD CARD	119
SAVE SCREEN SHOT TO SD CARD	119
SCROLL SCREEN AREA	120
SET AUX ESCAPE	120
SET AUX PORT AUTO READ	121
SET BAUD RATE	121
SET BAUD RATE (Output Prompt First)	122
SET BAUD RATE (Sticky and Prompt First)	122
SET COLOR (Basic)	123
SET COLOR (Detailed)	124
SET CONTROL PORT	124
SET CURSOR	125
SET DRAW MODE	125
SET FONT	126
SET or QUERY (LATCHING) STATE BUTTON	126
SET LED	127
SET ORIGIN	127
SET PEN WIDTH	127
SET PREVIOUS CONTROL PORT	127
SET TEXT ALIGNMENT	128
SET TEXT MODE	128
SET TOUCH CHARACTERISTICS	129
SET TOUCH DEBOUNCE	129
SET TYPEMATIC PARAMETERS	130
SET VARIABLE	130
SET VARIABLE (HEX)	131
SLIDER DEFINE	132
SLIDER VALUE	132
SPEAKER ON / OFF (Rev D and later boards only)	133
SPEED TEST	133
SPLASH SCREEN	133
TEXT DISPLAY	134

	TEXT FLASHING DELETE	136
	TEXT FLASHING DISABLE	136
	TEXT FLASHING DISPLAY	137
	TEXT FLASHING ENABLE	137
	TEXT FLASHING SYNCHRONIZE	138
	TOUCH CALIBRATE	138
	TOUCH MACRO ASSIGN	138
	TOUCH MACRO ASSIGN QUIET	139
	TOUCH MACRO ASSIGN WITH PARAMETERS	139
	TOUCHSCREEN ON/OFF	141
	TOUCH SIMULATE MOMENTARY	141
	TOUCH SIMULATE PRESS	141
	TOUCH SIMULATE RELEASE	142
	TOUCH SIMULATE TYPOMATIC	142
	UTF8 ENABLE / DISABLE	142
	VERSION	143
	WAIT	143
	WAIT FOR REFRESH	143
	WAIT VERTICAL RETRACE	144
	WINDOW RESTORE	144
	WINDOW RESTORE RECTANGLE	144
	WINDOW SAVE	145
6.	FONTS	146
6.1	EXTERNAL FONTS.....	146
6.2	CHARACTER SET – ISO 8859-1.....	147
7.	COLOR SPECIFICATIONS	148
7.1	OVERVIEW.....	148
7.2	RGB565 ENCODING.....	148
8.	USING CRC'D COMMANDS	149
8.1	OVERVIEW.....	149
8.2	COMMAND PROTOCOL.....	149
8.3	EXAMPLE CRC GENERATION CODE.....	149
9.	COMMUNICATIONS WATCHDOG TIMER	152
9.1	OVERVIEW.....	152
9.2	USING THE ‘*COMWDT’ COMMAND.....	152
9.3	EXAMPLE.....	153
10.	WORKING WITH VARIABLES	155
10.1	OVERVIEW.....	155
10.2	USER VARIABLES.....	155
10.3	SYSTEM VARIABLES.....	156
10.4	FORMATTING VARIABLES.....	157
11.	USING SIMPLE MATH EXPRESSIONS	159
11.1	OVERVIEW.....	159
11.2	LIMITATIONS AND REQUIREMENTS.....	159
11.3	EXAMPLES.....	160

APPENDIX A – LCD AND TOUCH PANELS COMPATIBLE WITH THE SLCD5+N CONTROLLER.....	161
A.1 LCD PANELS.....	161
A.2 RESISTIVE TOUCH PANELS (HARDWARE VERSION DEPENDENT)	161
A.3 PROJECTED CAPACITIVE TOUCH PANELS (HARDWARE VERSION DEPENDENT).....	161
APPENDIX B – PARTS AND SUPPLIERS FOR SLCD5+N CONNECTIONS	162
B.1 CONNECTORS AND CABLES FOR J6, J7, J13	162
B.2 CONNECTOR FOR J2	162
B.3 CONNECTORS FOR J11, J12	162
B.4 MATING CABLE FOR J10	162
B.5 DISCRETE WIRE CABLE VENDORS.....	162
APPENDIX C – ORDERING INFORMATION.....	163
C.1 GENERAL INFORMATION.....	163
C.2 CONTACT REACH DIRECTLY FOR SPECIFIC ORDERING INFORMATION	163
APPENDIX D – BMPLOAD PROGRAM	164
D.1 OVERVIEW	164
D.2 BITMAP COMPRESSION	164
D.3 BITMAP FORMAT.....	164
D.4 PROGRAM OPERATION.....	165
D.5 CONNECTING VIA SERIAL PORT	167
D.6 BMPLOAD SPEED ISSUES.....	167
APPENDIX E – MACRO COMMANDS AND FILE FORMAT	168
E.1 INTRODUCTION AND LIMITATIONS.....	168
E.2 MACRO FILE FORMAT.....	168
E.3 MACRO PARAMETERS (ARGUMENTS).....	170
E.4 ASSIGNING MACROS TO BUTTONS	170
E.5 SPECIAL MACRO COMMANDS, AND MACRO LABELS	170
<i>Memory commands</i>	<i>170</i>
<i>Special Arguments</i>	<i>171</i>
<i>Repeat Command</i>	<i>171</i>
<i>Labels</i>	<i>172</i>
E.6 CHANGING THE FIRMWARE POWER-ON BAUD RATE	173
E.7 CHANGING THE FIRMWARE POWER-ON BAUD RATE	173
APPENDIX F – TROUBLESHOOTING.....	174
F.1 TOUCH UNRELIABLE OR NON-OPERATIVE	174
APPENDIX G – EVALUATION / DEMO KIT TUTORIAL	175
G.1 SELF-RUNNING DEMONSTRATION	175
G.2 CONNECTION AND CONTROL VIA PC	175
G.3 SIMPLE COMMANDS.....	175
G.4 BITMAPS.....	176
G.5 MACROS	177
G.6 FONTS.....	177
G.7 ATTACHING TO THE EMBEDDED CONTROLLER	178

APPENDIX H – WORKING WITH BITMAPS	179
H.1 CREATING BITMAPS	179
H.2 HIGH COLOR	179
H.3 GRADIENTS	179
H.4 TRANSPARENCY	180
APPENDIX I – SD CARD SUPPORT FEATURES	181
I.1 OVERVIEW	181
I.2 FIRMWARE UPGRADE.....	181
I.3 HOLDING BINARY WORKING SET FILE.....	181
I.4 UPDATE BINARY FILE IN FLASH	181
I.5 SCREEN SHOT SAVING	182
I.6 RUN DEMO MACRO.....	182
I.7 MULTIPLE BINARY FILES	182
I.8 SEPARATE IMAGE FILES.....	182
APPENDIX J – LED BACKLIGHT DRIVER SCHEMATICS	183
APPENDIX K – BOARD VERSION DIFFERENCES.....	184
<i>Differences from SLCD5/SLCD5+ Products</i>	<i>184</i>
42-0323-001.....	184
42-0323-101.....	184
42-0323-301.....	184
APPENDIX L – MANUAL CHANGE HISTORY	185

Figures

FIGURE 1: 42-0323-001 AND 42-0323-101 CONTROLLER (FRONT)	15
FIGURE 2: 42-0323-001 AND 42-0323-101 CONTROLLER (BACK).....	15
FIGURE 3: 42-0323-301 CONTROLLER (FRONT)	16
FIGURE 4: 42-0323-301 CONTROLLER (BACK).....	16
FIGURE 5: SLCD5+N DIMENSIONS (INCHES).....	18
FIGURE 6: SLCD5+N CONNECTORS AND JUMPERS	19
FIGURE 7: TYPICAL (RS-232) CONNECTION FROM A PC	22
FIGURE 8: SLCD5+N POWER CONNECTIONS	31
FIGURE 9: SLCD5+N COMMUNICATIONS	32
FIGURE 10: SYSTEM SOFTWARE BOOT ALGORITHM.....	45
FIGURE 11: LED BACKLIGHT DRIVER SCHEMATICS	183

0. Hardware Limited Warranty and Software License Agreement

0.1 Hardware Limited Warranty

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

0.2 Returns and Repair Policy

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

Authorized returns must be shipped freight prepaid to Reach Technology Inc. 4575 Cushing Parkway, Fremont, California 94538 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 USING THE SOFTWARE.

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

BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, DO NOT INSTALL OR USE THE SOFTWARE. IF YOU HAVE PAID A FEE FOR THIS LICENSE AND DO NOT ACCEPT THE TERMS OF THIS AGREEMENT, REACH WILL REFUND THE FEE TO YOU PROVIDED YOU (1) DO NOT INSTALL THE SOFTWARE AND (2) RETURN ALL SOFTWARE, MEDIA AND OTHER DOCUMENTATION AND MATERIALS PROVIDED WITH THE SOFTWARE TO REACH TECHNOLOGY INC AT: REACH TECHNOLOGY INC., 4575 Cushing Parkway, Fremont, California 94538.

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

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

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

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

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

HIGH RISK ACTIVITIES. The Software Product is not fault-tolerant and is not designed, manufactured, or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software Product, or any software, tool, process, or service that was developed using the Software Product, could lead directly to death, personal injury, or severe physical or environmental damage (“High Risk Activities”). Accordingly, Reach and its suppliers and licensors specifically disclaim any express or implied warranty of fitness for High Risk Activities. You agree that Reach and its suppliers and licensors will not be liable for any claims or damages arising from the use of the Software Product, or any software, tool, process, or service that was developed using the Software Product, in such applications.

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. Introduction

1.1 Overview

This manual describes the SLCD5+N controller. It does not cover the older SLCD5 or SLCD5+ controllers, which have their own manual.

The SLCD5+N controller provides complete a Graphical User Interface for embedded systems using VGA, WVGA, and SVGA panels. Using the SLCD5+N is simply the quickest way to generate a user interface without a lot of graphical programming. It has a small size to fit in space-constrained applications.

1.2 Controller Features

- Drives LCD TFT displays with VGA, WVGA, or SVGA resolution; orderable options for panel interface includes:
 - 3.3V CMOS level compatible interface (parallel RGB).
 - LVDS interface.
- 16-bit color (565 mapping – same as PC in 16 bit mode or Mac in thousands of color).
- Touch controller (4 wire resistive, or I2C projective-capacitive).
- Beeper for audible touch feedback and alarms.
- 3” by 4.5” size, 0.55” thick .
- High speed ARM9 processor (266 MHz).
- On-board RS232, RS485, and CMOS-level interfaces, up to 460,800 baud.
- Up to 28MB highly reliable NOR data flash for user downloadable bitmaps with optional RLE compression.
- On-board LED backlight driver as an orderable option
- Backlight enable and brightness control.
- SD card slot for firmware upgrades and bitmap / macro storage.
- Can be modified for specific OEM requirements.

1.3 Board Pictures

42-0323-001, 42-0323-101

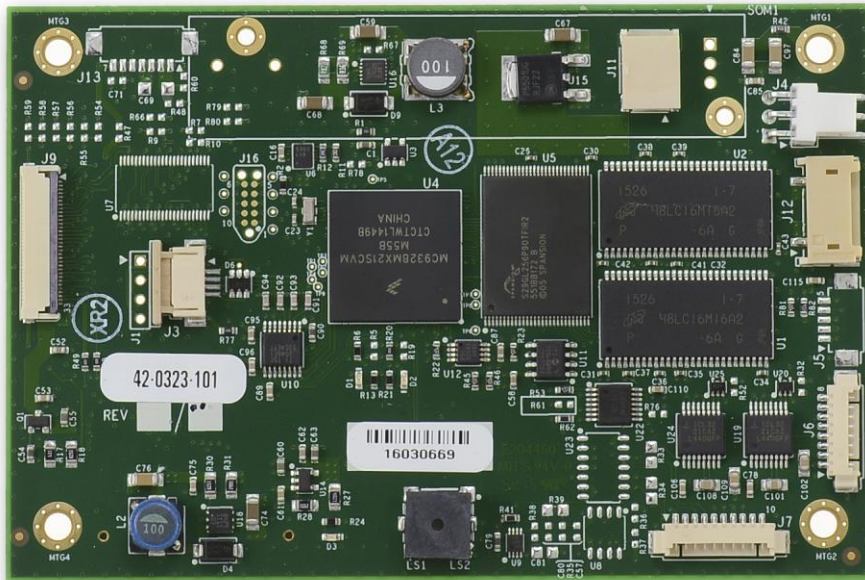


Figure 1: 42-0323-001 and 42-0323-101 Controller (Front)

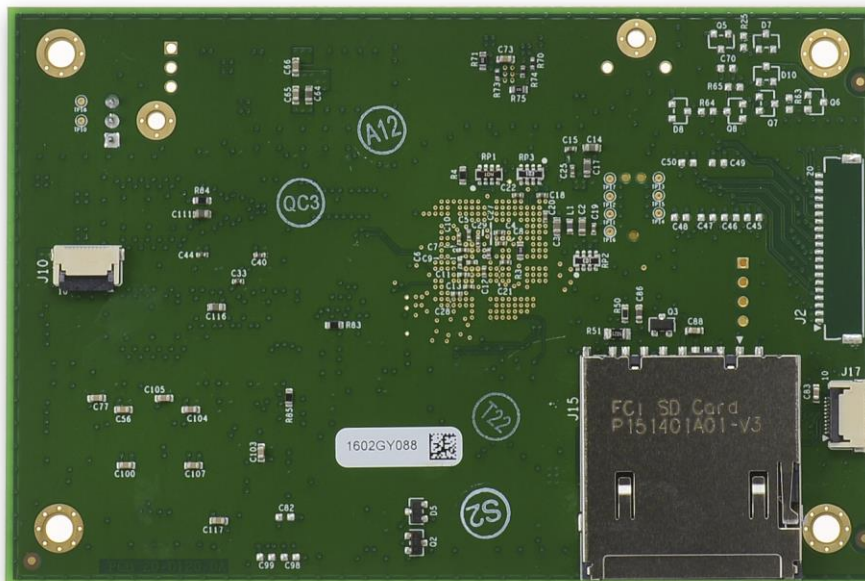


Figure 2: 42-0323-001 and 42-0323-101 Controller (Back)

Note: The only difference between the -001 and the -101 controllers is the backlight current, as determined by resistor R68, which is not installed on the -001 and is installed on the -101 (shown). See [Appendix J](#) and [Appendix K](#) for additional information.

42-0323-301

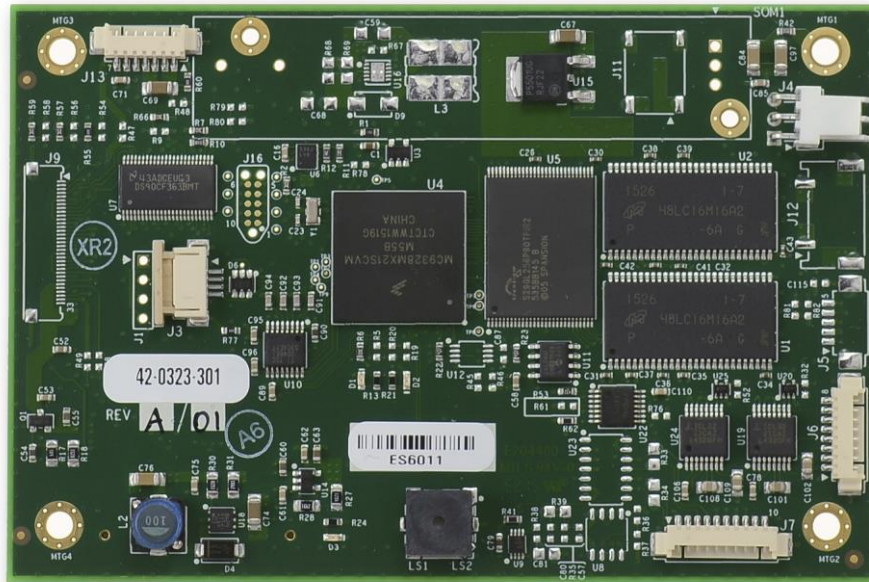


Figure 3: 42-0323-301 Controller (Front)



Figure 4: 42-0323-301 Controller (Back)

1.4 Ordering Information

Contact Reach Technical Support for orderable part numbers. These vary by option and type of LCD panel being driven.

1.5 Electrical Characteristics

Note: If you have a Reach Display Module that uses the SLCD5+N, you need to refer to the Data Sheet for the Reach Display Module you are using for correct electrical characteristics (typically, the Display Module input voltage is more restrictive due to the LCD panel requirements).

If you are integrating the SLCD5+N Controller Board with your own display panel, you must determine the appropriate power for your display.

The SLCD5+N Controller Board has two power inputs – “main” and “backlight”. The main can take 5-12V DC +/- 10%. An on-board switching regulator generates the 3.3V required by the LCD panel and the SLCD5+N processor. The backlight power is used either for pass-through to an external LED driver (may be in the LCD), or for the optional on-board LED backlight driver. The on-board driver is a “boost” type and ***must be*** powered by a voltage less than the panel’s LED input typical working forward voltage. For example, if the backlight LED typical voltage is 9.9V, the backlight input voltage is typically 5V.

LCD panels that require 3.3V power and have a 3.3V “CMOS” level compatible data interface are supported, as are LCD panels with an LVDS interface. A list of compatible panels is provided in [Appendix A](#).

The power requirements shown below are typical values. The theoretical absolute worst case is up to 2 times these values, but it is extremely unlikely that this will occur on a specific board. We suggest a 150% of the below values is a reasonable value.

The following table gives the SLCD5+N board only power supply requirement; for a total power budget you must include the input-referenced panel power plus backlight power.

Supply Voltage:	Typical Current (beeper off):	Typical Current (beeper on full):	Typical Current (Vin -10% , speaker option, max volume):
5V nominal	300mA (typ.)	364mA (typ.)	546mA (typ.)
12V nominal	143mA (typ.)	204mA (typ.)	247mA (typ.)

Backlight

If you need to calculate the LED backlight driver power efficiency, see [Appendix J](#) for the backlight driver schematics.

1.6 Panel Support

See [Appendix A](#).

1.7 SD Card Support

The SD card slot supports cards that are FAT 16 formatted, maximum 2GB. We have tested, and recommend SanDisk brand cards. You may need to format them on a PC with FAT or FAT16 format selected before use (DO NOT USE “FAT32”).

Long filenames are not supported; only 8.3 format names can be used.

See also [Appendix I](#).

1.8 Dimensions

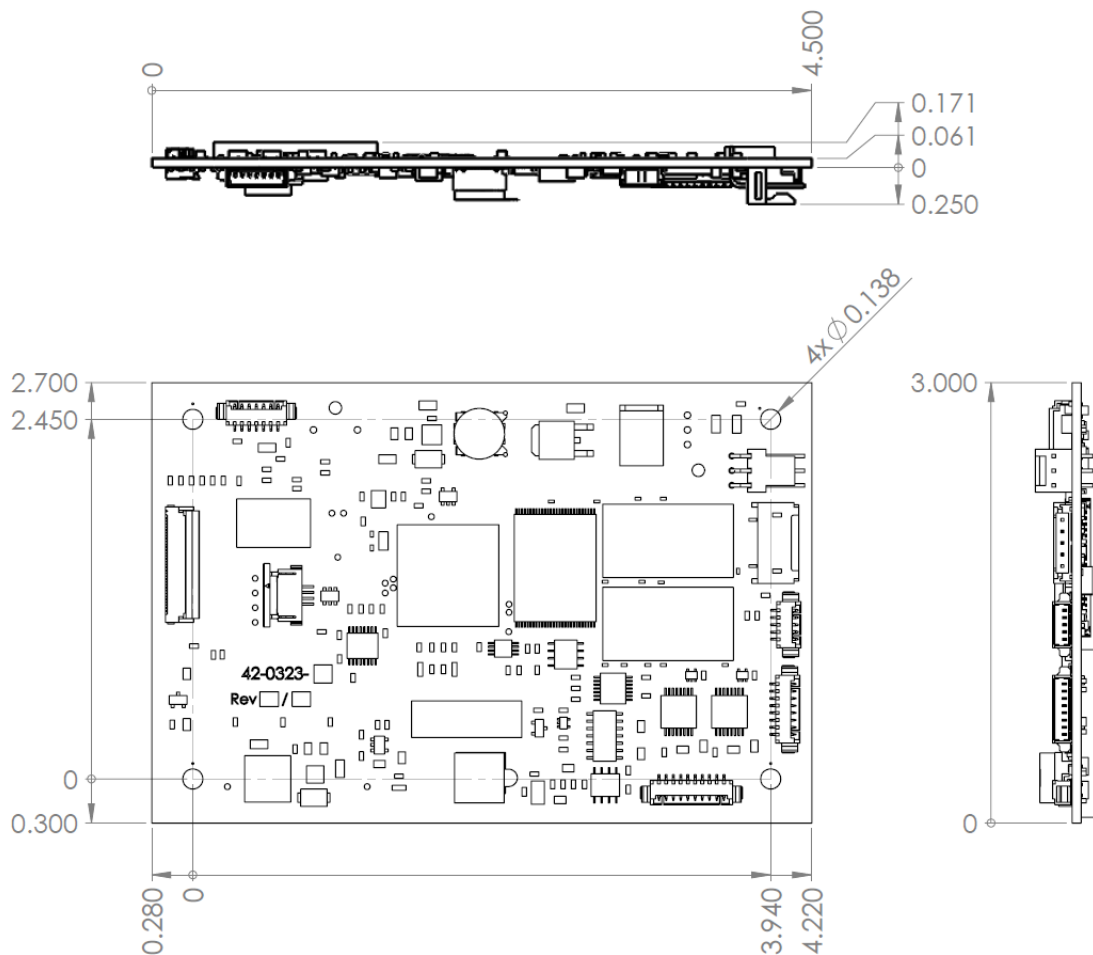


Figure 5: SLCD5+N Dimensions (Inches)

2. Connectors and Jumpers

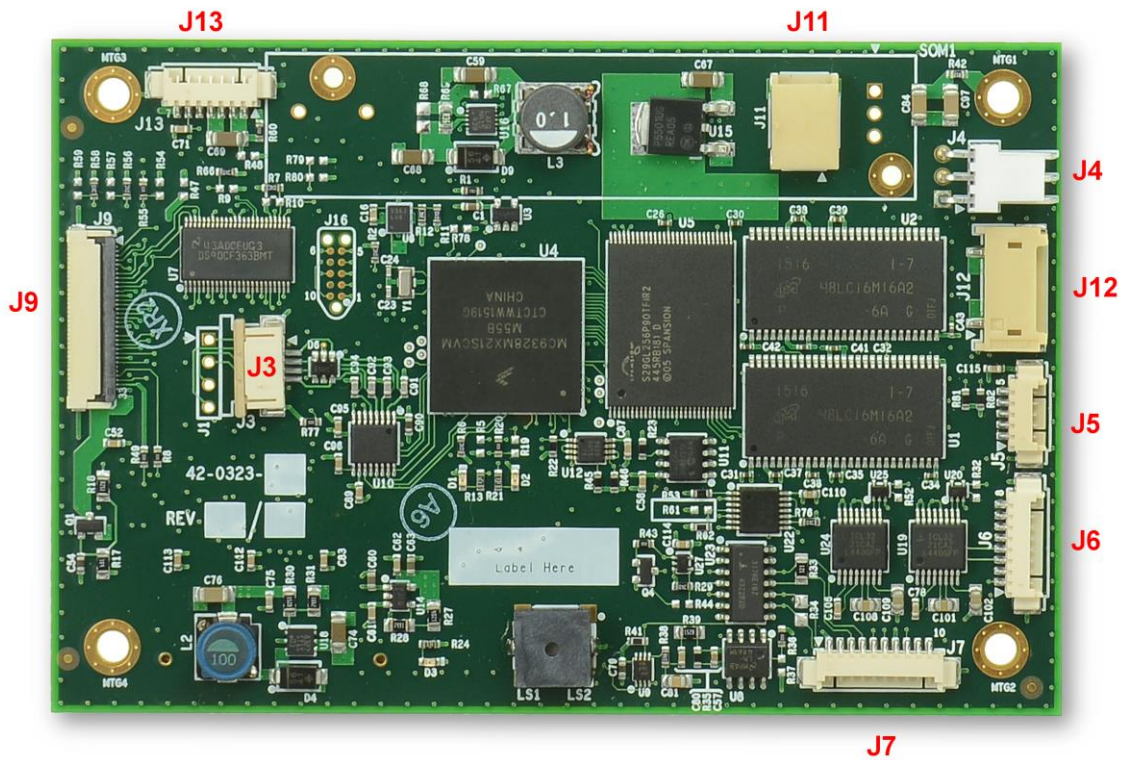


Figure 6: SLCD5+N Connectors and Jumpers

Not shown, on back of board:

- J2 – LVDS connector
- J10 – I2C PCAP connector
- J15 – SD card slot
- J17 – I2C PCAP connector

2.1 Overview

The SLCD5+N requires the following major connections for operation: **Power, Communications, Panel, Touch, and Backlight**. The following table summarizes these connections and options.

Please note that all signals named Ground are connected to the board mounting holes and must connect to chassis ground for ESD protection.

Purpose	Use	Notes
POWER	J6 – or – J4	J6 has both power and communications signals. Alternate power connector. Can also be used as power out if J6 is providing power. Required if inverter (see below) takes more than 1A current.
COMMUNICATIONS	J6 J7	COM0 port (RS-232 or 3.3V CMOS). COM1 port (RS-232, RS485/RS422, or 3.3V CMOS).
PANEL	J9 – or – J2	3.3V CMOS interface (-001, -101). LVDS panels (on back of board) (-301).
TOUCH	J1, J3 – or – J10, J17	4-wire resistive. I2C PCAP (on back of board) (-001, -101).
EXT. BACKLIGHT	J13	Not installed on -001, -101. -301: analog and PWM dimming options supported.
BACKLIGHT	J11, J12	Backlight power. Set to 140ma (-001) or 200ma (-101). Not installed on -301.
AUDIO	J5	Audio output (not installed).

2.2 J6 – Power and Communication COM0 (RS232 Mode)

On the standard SLCD5+N controller, J6 supports either RS-232 levels OR 3.3V CMOS levels as a serial port interface. The interface mode is automatically selected by the applied signal. Connect a valid RS232 signal to pin 3 to select the RS232 mode.

J6 8 Pin Molex 53261-0871 for Power and Main Communications

Pin	RS232 Mode – Selected by applied signal on attached cable
1	Do not connect
2	Do not connect
3	RS232 input (ESD protected 15KV)* Valid level enables RS232 mode
4	RS232 output (ESD protected 15KV)*
5	Backlight power input, feeds directly to J13, max. 1A (Note 2, 3)
6	Ground
7	5 – 12V main power Input (Note 1, 3)
8	Ground

Note 1: The board can be powered either through J6 OR J4. If it is powered through J4, pins J6-5 and J6-7 do not need to be connected.

Note 2: If pin 5 is powered with 5V for a 5V backlight driver / inverter, resistor R8 (zero ohms) needs to be installed for full speaker volume.

Note 3: Consult appropriate Reach Display Module Data Sheet; some modules work only with 5V, some only with 12V.

**ESD protection via ICL3222E ESD protected RS-232 transceiver or equivalent.*

COM0 Cable

The SLCD5+N Development Kit includes a ‘Y’ cable that connects to connector J6. This cable provides a barrel connector for power (5V or 12V, depending on model) and a female DB9 connector for the RS232 connections on connector J6. The DB9 connector is wired to connect directly to an RS232 port on a PC, or on a USB-RS232 adaptor.

The 5V cable is Reach P/N 23-0178-18. This cable is included with Development Kits for 5V display modules.

The 12V cable is Reach P/N 23-0181-18. This cable is included with Development Kits for 12V display modules.

Typical connection from a PC (RS-232) is as follows:

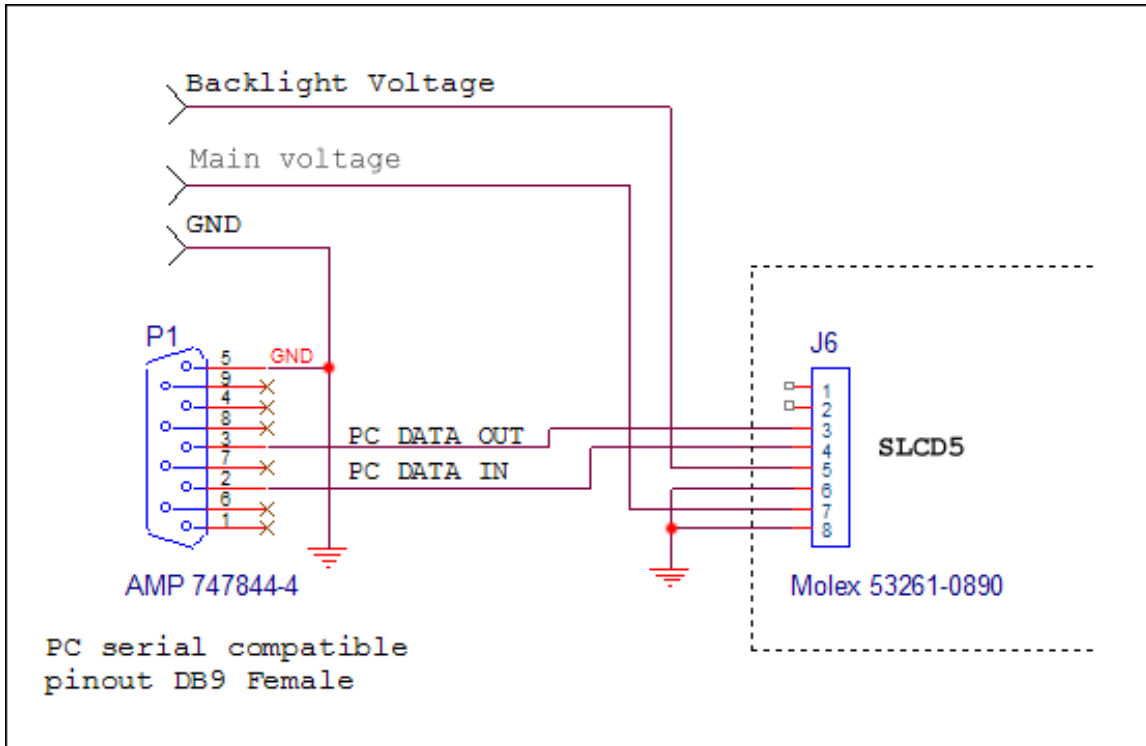


Figure 7: Typical (RS-232) Connection from a PC

2.3 J6 – Power and Communication COM0 (3.3V CMOS Mode)

To use this mode, omit the RS232 connections on pins 3 and 4, and use pins 1 and 2 instead. The absence of a valid RS232 signal on pin 3 selects 3.3V CMOS mode for COM0.

J6 8 Pin Molex 53261-0871 for Power and Main Communications

Pin	CMOS I/F Mode – Selected by wiring harness
1	UART TxD (output from SLCD5) (Note: Connects directly to CPU!)
2	UART RxD (input to SLCD5) (Note: Connects directly to CPU!)
3	Do not connect (the absence of this signal selects 3.3V CMOS mode)
4	Do not connect
5	Backlight power input, feeds directly to J13, max. 1A (<i>Note 2, 3</i>)
6	Ground
7	5 – 12V main power Input (<i>Note 1, 3</i>)
8	Ground

Notes: See Section 2.2 above.

2.4 J7 – Communication COM1 and Reset

On the standard SLCD5+N controller, J7 supports either RS-232 levels OR 3.3V CMOS levels as a serial port interface. This is selected by the attached cable.

To select RS232 mode, connect valid RS232 signals to pin 6. J7 can also support full duplex RS485/RS422 as a manufacturing option.

To select 3.3V CMOS mode, omit the RS232 connections on pins 5 and 6, and use pins 4 and 10 instead. The absence of a valid RS232 signal on pin 6 selects 3.3V CMOS mode for COM1.

J7 10 Pin Molex 53261-1071 for RS232 COM1 Communications

Pin	Connections for RS232 Mode
1	3.4V output (for powering external transceiver)
2	No connect
3	No connect
4	No connect
5	RS-232 out (ESD protected 15KV*)
6	RS-232 in (ESD protected 15KV*) Valid level enables RS232 mode
7	Ground
8	RESET- input (active low with on-board pull-up)
9	Not used
10	No connect

*ESD protection via ICL3222E ESD protected RS-232 transceiver or equivalent.

Example J7 Cable for connecting to PC serial port via DB9-Female

J7 Pin	DB9 (Female) pin	Signal
5	2	RS-232 from SLCD5+N to PC
6	3	RS-232 from PC to SLCD5+N
7	5	Ground

COM1 Cable

The SLCD5+N Development Kit includes a cable that connects to connector J7. This cable provides a DB9 connector for the RS232 connections on connector J7. The DB9 connector is wired to connect directly to an RS232 port on a PC, or on a USB-RS232 adaptor.

The Reach P/N for this cable is 23-0182-12. This cable is included with all SLCD5+N Development Kits.

J7 10 Pin Molex 53261-1071 for 3.3V CMOS COM1 Communications

Pin	Connections for 3.3V CMOS Mode
1	3.4V output (for powering external transceiver)
2	RTS (can be used to enable external driver) buffered, 3.3V CMOS levels, 5v tolerant
3	No connect
4	RxD- (standard UART receive data input), buffered, 3.3V CMOS levels, 5v tolerant
5	Do not use
6	Do not use (the absence of this signal selects CMOS mode)
7	Ground
8	RESET- input (active low with on-board pull-up)
9	Do not use
10	*TxD- (standard UART transmit data output) buffered, 3.3V CMOS levels, 5v tolerant

J7 10 Pin Molex 53261-1071 for RS485/RS422 COM1 Communications *NOTE: THIS IS AVAILABLE ONLY AS A MANUFACTURING OPTION.*

Pin	Connections for RS485 / RS422 Mode
1	3.4V output (for powering external transceiver)
2	Inverting driver output*
3	Inverting input**
4	Non-inverting input**
5	Do not use
6	Do not use
7	Ground
8	RESET- input (active low with on-board pull-up)
9	Do not use
10	Non-inverting driver output*

*On-board optional output termination resistor is R34.

** On-board optional input termination resistor is R33.

2.5 J4 – Power

Used if separate power input is desired. Can also be used as pass-through output for power supplied via J6.

J4 3 Pin Molex 22-05-3031 0.1” polarized or equivalent

Pin	Signal
1	Backlight and Speaker regulator power. Connects directly to J6-5, J13-1, J13-2. Typically 5V or 12V (<i>Note 1,2, 3</i>) DO NOT use 12V if the on-board LED driver option is provided.
2	Main power, connects directly to J6-7. Typically 5V or 12V (<i>Note1, 3</i>)
3	Ground (common to main and inverter power)

Note 1: The board can be powered either through J6 OR J4. If it is powered through J4, pins J6-5 and J6-7 do not need to be connected.

Note 2: If pin 1 is powered with 5V, resistor R8 (zero ohms) can be installed for full speaker volume.

Note 3: Consult appropriate Reach Display Module Data Sheet; some modules work only with 5V, some only with 12V.

2.6 J9 – 33 pin 0.5mm Flat Flex (hardware version dependent)

(Note: Contacts are both top and bottom – verify cable orientation before applying power.)

J9 33 Pin Omron XF2M-3315-1 0.5mm pitch Zero-Insertion-Force Connector

Pin	Signal	Pin	Signal
1	GND	18	Green 5
2	LCD Clock	19	GND
3	LCD Line Pulse (HSYNC)	20	Blue 0 (= Blue 5)
4	LCD Frame Pulse (VSYNC)	21	Blue 1
5	GND	22	Blue 2
6	Red 0 (= Red 5)	23	Blue 3
7	Red 1	24	Blue 4
8	Red 2	25	Blue 5
9	Red 3	26	GND
10	Red 4	27	LCD DE (Data Enable)
11	Red 5	28	LCD VCC (3.3V)
12	GND	29	LCD VCC (3.3V)
13	Green 0	30	R/L (Note 1)
14	Green 1	31	U/D (Note 1)
15	Green 2	32	V_Q or backlight power (Note 2)
16	Green 3	33	GND
17	Green 4		

Note: The SLCD5+N supports 565 16-bit color, so the least significant red and blue color bits are set to the most significant bit. This preserves dynamic range at the expense of middle level steps

*Note 1. These signals can be set via SLCD5+N “*orient” command.*

Note 2. This pin can be either the V_Q signal or the backlight driver output or the backlight voltage per factory options. The V_Q signal is set by the board’s firmware and is panel-dependent.

2.7 J2 – LVDS LCD Panel (hardware version dependent)

J2 DF19G-20P-1H(54)

Pin	Signal
1	LCD VCC (3.3V)
2	LCD VCC (3.3V)
3	GND
4	GND
5	TXOUT0_N
6	TXOUT0_P
7	GND
8	TXOUT1_N
9	TXOUT1_P
10	GND
11	TXOUT2_N
12	TXOUT2_P
13	GND
14	TXCLK_N
15	TXCLK_P
16	GND
17	V_Q / backlight – consult factory
18	OPT1 – consult factory
19	OPT2 – consult factory
20	OPT3 – consult factory

The V_Q and OPT1-3 pins are typically configured per the specific LCD panel being driven. Some LCD panels need static signals to select 18 bit vs 24 bit mode and LVDS mapping. Some LCD panels include LED driver power on this connector.

2.8 J3 – 4 Wire Touch

J3 4 Pin Molex 52271-0469 or equivalent bottom contact 1mm pitch bottom contact Zero-Insertion-Force

Pin	Signal*
1	X Right / X+
2	Y Down / Y+
3	X Left / X-
4	Y Up / Y-

*Connector compatible with Fujitsu N010-0554-T511 8.4" 4-wire touch or similar.
All signals are ESD protected via California Micro Devices PACDN044Y5.
Signal assignment is not important as long as the order from 1 to 4 is "around the panel"
as calibration will compensate for any X / Y swap and + / - swap.*

2.9 J1 – 4 Wire Touch

J1 4 Pin 0.1" pitch 0.025" square post header

Pin	Signal
1	X Right / X+
2	Y Up / Y+
3	X Left / X-
4	Y Down / Y-

2.10 J10, J17 – PCAP I2C Touch Sensor (hardware version dependent)

J10 10 Pin Omron XF2M-1015-1A, for 7" Evervision VGG804808-6UFLWL.

J17 10 Pin Omron XF2M-1015-1A, for 5.7" Evervision VIM05700.

Pin	Signal
1	Ground
2	N/C
3	N/C
4	N/C
5	SCL_EXT
6	SDA_EXT
7	PCAP_INTN
8	PCAP_RST
9	+3.15V
10	Ground

J13 – Backlight / Inverter Control (hardware version dependent)

J13 7 Pin Molex 53261-0771

Pin	Signal
1	Backlight power (connects directly to J6-5, J4-1)
2	Backlight power (connects directly to J6-5, J4-1)
3	Ground
4	Ground
5	Backlight on/off control, 0V – 5V, see Note
6	Backlight brightness control (analog voltage) 0 – 5V, see Note
7	PWM brightness, 0V – Backlight voltage, see Note

Note: This connector is used to power and control the panel backlight driver. The active sense of pins 5-7 (active high or low) is set by software command. The range and polarity of pins 6 and 7 are also set by software command.

2.11 J11, J12 – On-board Backlight Driver (hardware version dependent)

J11 JST SMO2B-BHSS-1-TB(LF)(SN)

J12 JST SM02(8.0)B-BHS-1-TB(LF)(SN)

Pin	Signal
1	+V, current regulated LED backlight output
2	Ground

2.12 J16 – JTAG

The JTAG connector is for factory use only.

2.13 J5 – External Audio Output (hardware version dependent)

J5 5 Pin Molex 53261-0571 or equivalent

Pin	Signal
1	Audio out (-)
2	Audio out (+) / +5V
3	Ground
4	Speaker out 1
5	Speaker out 2

The Audio out provides a square wave at audio frequency and is the same signal as connected to the on-board beeper. The Speaker output is designed to connect to an 8 ohm speaker, typically 0.5W – 1W size. The speaker vs. beeper selection is made by firmware command (see [SPEAKER ON/OFF](#)).

2.14 J15 SD Card Connector

The SD card connector on the back of the SLCD5+N is compatible with standard SD cards.

J15 ALPS P/N SCDA\$A0301

Pin	Signal
1	SD_DAT3
2	SD_CMD
3	GND
4	V_CARD (approx. 3.15V)
5	SD_CLK
6	GND
7	SD_DAT0
8	SD_DAT1
9	SD_DAT2
10	GND (Tab)
11	GND (Tab)
12	GND (Tab)
13	n/c
14	SD_DETECT
15	SD_WP
16	GND

3. Configuration Guide

3.1 Power Connections

The diagram below illustrates the power flow in an SLCD5+N configuration.

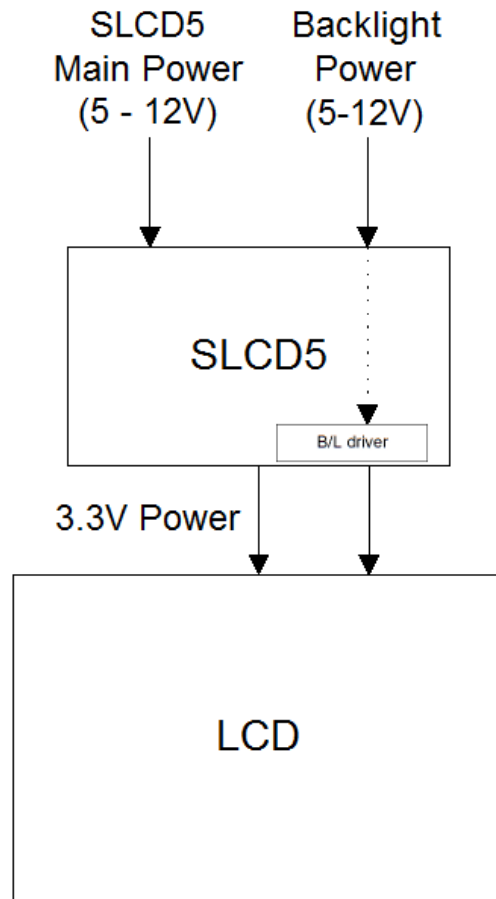


Figure 8: SLCD5+N Power Connections

3.2 Serial Ports

The SLCD5+N can use either RS-232 levels or CMOS logic levels for serial communication. In the standard configuration, the COM0 and COM1 serial ports are configured for either RS-232 or 3.3V CMOS, as selected by the attached cables. COM1 can be configured as RS485/RS422 full duplex as a manufacturing option – consult the factory.

In the SLCD5+N development kits, the COM0 and COM1 ports are referred to as the “Main” and “Aux” ports respectively. The Main port is connected to an embedded processor “host” and controls the display. The Aux port is typically connected to a PC and is used to update the bitmaps and macros stored on the board and to test commands.

By default, the serial communication protocol is 115200 baud, 8 data bits, no parity, with 1 stop bit, and software (XON/XOFF) flow control. The baud rate can be changed by using a power-on macro (see [Appendix E](#)), or by using the [SET BAUD RATE \(Sticky\)](#) command.

The following diagram illustrates this communications setup.

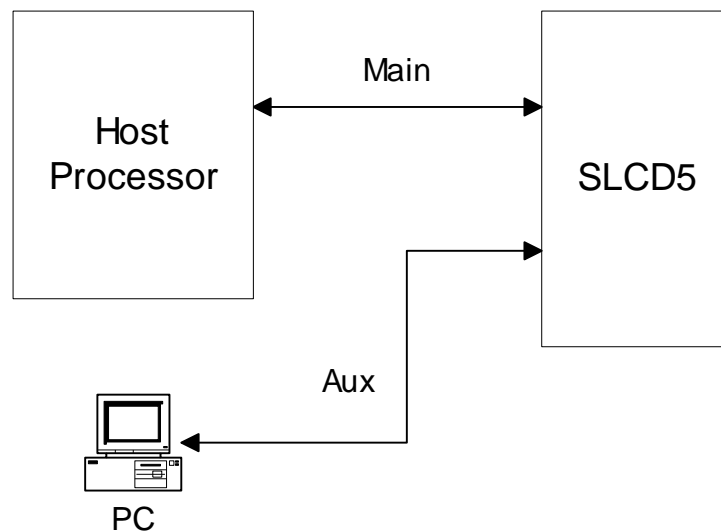


Figure 9: SLCD5+N Communications

3.3 TFT Hardware Panel Orientation

Some TFT LCD panels have orientation signals such as “UP / DOWN”, and “LEFT / RIGHT” that allow the display orientation to be flipped in the horizontal and / or vertical orientation. The SLCD5+N provides these signals on the LCD connector, and they can be set under software control using the [*orient](#) command or the [orient](#) config.ini setting. You may wonder why there is a left / right which makes the display unreadable. In some applications a mirror is used to make the display system more compact and since a mirror reverses the display, pre-reversing it makes it show as expected.

3.4 System Configuration

The SLCD5+N maintains touch calibration and option parameters in non-volatile memory (EEPROM). These parameters include LCD timing, backlight interface details, startup command, and so on. These can be set via the serial command interface, but in a production environment, it is recommended to set them via a file on an SD card. In this way, the SLCD5+N can be configured for production simply by inserting the appropriate SD card and cycling power. When the controller powers up, it checks for an SD card, and if present looks for and reads the configuration file and stores the specified values into EEPROM. This is done only once, as typically the SD card is not shipped installed in the unit.

Warning: The EEPROM has a limited life of 1,000,000 writes. The CONFIG.INI file causes the EEPROM to be written, and so it should not be present on an SD card installed in a production system.

The config.ini file must be in the root directory of the SD card. It is a plain text file, with one or more lines in the following format:

```
<variable_name> = <value>
```

Comments are indicated by a # character at the beginning of a line. Empty lines are allowed and ignored.

Note that unlike a PC Windows system, the config.ini file only needs to be present once on power-on to set the board optional parameters. It should not be present on an SD card that is present in a production unit since it will re-write the EEPROM each time the board is powered on.

CONFIG.INI Example:

```
#---- start of file-----
#this is an example

#display active config.ini statements on COM0 port at
power-on
verbose = 1

#set power on macro to 1
PONMAC = 1
#PONMAC = 2 - example of how to comment out an
alternate value

#set test string response to *config command
config = "test config file"

#---- end of file ----
```

CONFIG.INI Options: (* indicates default)

Variable Name	Values	Action
verbose	0* or 1	If 1, the valid config variables are displayed on the console port at power-on. This helps in debugging or validating a config.ini file.
mainPort	0* or 1	Sets the COM port that is active on power-on. Note that boot and startup messages will always be displayed on COM0.
orient	0..3	Equivalent to *orient command.
invEnaHiTrue	0 or 1	Same as *invEnaHiTrue command.
pwmlsEnable	0 or 1	Same as *pwmlsEnable command.
maxBrite	0..255	Same as *maxBrite command.
minBrite	0..255	Same as *minBrite command.
brite	0..255	Same as xbb command.
touchSwap	0 or 1	Same as *touchSwap command This is no longer needed due to 5 point calibration.
tsamp	0..20	Default value is panel-dependent. See *touchParm command.
tspan	0..20	Default value is panel-dependent. See *touchParm command.
touchMode	0*..7	bit 1 => lockout bit 2 => validate touch pressure bit 3 => touch wander acts as release See *touchMode command.
tplo	0..65535	Default value is panel-dependent.
tphi	0..65535	Default value is panel-dependent.
tcTimeout	0..99	Same as *tcTimeout command.
auxEsc		Same as *auxEsc command.
beepFreq	0..3000	Same as bf command.
beepVol	0..255	Same as bv command.
debounce	50..200	Same as *debounce command.
typematicDelay		Same as first argument of typematic command.
typematicRepeat		Same as second argument of typematic command.
PONMAC		Same as *PONMAC command , except that the <option> parameter is handled automatically. If macro <index> is 0 or 255, the copyright display is enabled, otherwise it is disabled.
SPL		Same as *SPL command.
hfp		Set LCD horizontal "front porch" timing in clocks.

hpw		Set LCD horizontal sync width timing in clocks.
hbp		Set LCD horizontal “back porch” timing in clocks.
vfp		Set LCD vertical “front porch” timing in lines.
vpw		Set LCD vertical sync width timing in lines.
vbp		Set LCD vertical “back porch” timing in lines.
config	“text string”	Sets response for *config command.
LCDshiftclock	0 or 1	0 = Data stable on positive edge of LCD Shift Clock. 1 = Data stable on negative edge of LCD Shift Clock.
baud0 baud1	460800 230400 115200 57600 38400 19200 9600 4800 1200	Set COM0 baud rate. Set COM1 baud rate. The new rate becomes active at the next power cycle.
externalSpeaker	0* or 1	0 = External Speaker OFF, on-board Beeper ON 1 = External Speaker ON, on-board Beeper OFF
autoCal	0* or 1	0 = autoCal disabled. 1 = autoCal enabled: if the touch screen is touched while the unit powers on, it will perform a touch calibration and then continue normal startup.
macdebug	0* or 1	Same as *macdebug command . Use to debug power on macro.

4. System Overview

4.1 General SLCD5+N Controller Information

The SCLD5+N acts as a “smart terminal” and is meant to be connected to a host processor that implements the desired Graphical User Interface (GUI) by issuing commands to the SLCD5+N and processing button press responses from the SLCD5+N. In this manual, the term “host” is used to describe the device connected to the SLCD5+N.

For use in systems where the connection between the SLCD5+N and the host needs to be checked for validity on a regular basis, a Communications Watchdog feature can be enabled to cause special messages to be sent to the “host” if the SLCD5+N does not receive input from the “host” for an interval determined by the host (see [COMMUNICATIONS WATCHDOG](#)).

4.2 Compatibility with SLCD, SLCD6, SLCD43, SLCD+ controllers

The SCLD5+N is generally software-compatible with the smaller screen SLCD family controller. The significant differences are:

- The built-in fonts are different. The SLCD fonts were specifically designed for a small screen. The SLCD5+N fonts are Microsoft Windows compatible to ease graphic bitmap development. The SLCD font names are still supported so that macros will not fail, but the fonts are different.
- The latching button response has changed; there is now a space between the button index and the state:
- Response format: `s<index> <state>`
Example: `s128 1`
- The SLCD5+N is much faster; this may impact host timing.

4.3 Bitmaps and Macros

To implement a GUI, a set of graphic images are needed for backgrounds, logos, buttons, switches, and so forth. These are created on a PC as bitmaps (.bmp files) in 24-bit color. Photoshop and Gimp can be used as bitmap editors. The bitmaps are compressed into a binary file which is then either downloaded directly into the SLCD5+N flash memory via the serial port, or stored on an SD card that is then inserted into the SLCD5+N.

This binary file can also contain macros and extra fonts. A macro is a set of commands that can be invoked with a single command, and this is detailed in [Appendix E](#).

In a development environment, the SD card is a convenient way to load bitmaps stored macros. Once development is complete, the SD card is also used to update the on-board flash memory. Once updated, an SD card is no longer needed to be shipped with the controller.

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.

See [Appendix D](#) regarding the *BMPLoad* program that is provided to store bitmaps and other data into the SLCD5+N.

4.4 Overview – SLCD5+N Development Kits

The SLCD5+N is available in a development kit form. It comes pre-loaded with bitmaps and macros that implement a demo if the unit is powered on with an SD card installed that contains a file “rundemo.ini”. This file is a standard text file that contains the line:

- `demomac = <macro_number>`

For example:

- `demomac = 1`

Where the `<macro_number>` is the macro to run at power-on, typically 1. The SLCD5+N development kit comes with DB9 adapter cables that make it easier to develop applications. One cable, connected to COM0 (J6), connects to the host processor and the other cable, connected to COM1 (J7), connects to a PC which is used to interactively develop screen content and test commands and also to download bitmap images.

4.5 Getting Started

The SLCD5+N kit as shipped contains a demo that allows you to verify its functionality and see various interface possibilities. Just plug the supplied power supply into the barrel connector on the COM0 cable. The display should light up and lead you through various touch-activated screens. If for some reason the touch is not working, you may need to run the [TOUCH CALIBRATE](#) command.

Note that in a development kit, the demo is preloaded into the unit’s flash memory, and includes both bitmap files and a macro file. To best learn how the SLCD5+N board and this kit works, start with simple commands using the serial interface and leave the creation and use of macros for later.

[Appendix G](#) provides a brief tutorial.

4.6 Connecting the development kit to a PC

The development kit should be connected to a PC so that the serial command interface can be experimented with. This is a preliminary step before the unit is connected to the embedded system that will control the kit.

As shipped, the serial port is set to 115,200 baud, 8-bit, 1 start, 1 stop, no parity. There are two cables with DB9 connectors. One cable (the “COM0 cable”) is for COM0 and power, and connects to connector J6. This is the “Main” comm port. The other cable (the “COM1 cable”) is for COM1 and connects to connector J7. This is for the “Aux” port.

Connect the PC using a straight through serial cable to the COM0 cable. A USB-to-serial adapter cable can also be used and plugged directly into this connector. Reach recommends the FTDI series of serial adapters, see www.ftdichip.com. These can be purchased from www.digikey.com. **Warning: The Belkin USB-serial adapter has software compatibility issues and is not recommended.**

Once connected, use a terminal emulator such as Realterm (<http://realterm.sourceforge.net/>) or Tera Term (<https://tssh2.osdn.jp/index.html.en>) to send and receive commands from the unit.

[Appendix G](#) provides a brief tutorial.

4.7 Switching Between PC and Embedded Controller

The SLCD5+N firmware has the concept of a “Main” control port and an “Aux” port. The main port is used to send commands to the unit. During development, it is useful to be able to switch which physical port is considered Main. This is useful so that the PC can take control and download new images, or be used to test a command via a serial terminal emulator.

Switching happens when the aux port receives three <return> characters in a row. After that, it switches to become the main port. The BMPload program uses this method to take control and download new bitmaps. It also uses the [*prevCons](#) command to switch back so that the embedded host has control.

The embedded host code should be written such that when it starts and tries to connect to the SLCD5+N, it issues a <return>, waits for the SLCD5+N prompt (2 characters, ‘>’ followed by <return>), and then tries again. Doing this, it will end up sending the three <return> characters in a row and auto switch the port it is connected to the main port.

NOTE: Only one port is in control at any given time.

See also [CONTROL PORT AUTOSWITCH](#).

4.8 Development Kit Operational Notes

1. The unit default baud rate is 115200. The unit does not echo characters (for communications efficiency), so you must select “echo characters locally” or set “half duplex” in your PC communications program. Also, all return strings are terminated by a <return> only, so you need to specify “add line feed to line return” as well.
2. The internal demo starts with an optional touch calibration. In order for the touch screen to work reliably, ensure the LCD frame is grounded to the SLCD5+N mounting holes.
3. The demo requires a certain set of bitmaps, macros, and fonts to be loaded. If these are not present, the demo will not run correctly. Copies of these are provided in the “Reach_SD_Card/<screen type>” folders on the CD provided. Copy the appropriate folder to the SD card, and install the SD card into the SLCD5+N to run the demo.
4. The COM0 cable provides the connection to the “MAIN” RS232 serial port. It connects to J6 of the SLCD5+N controller. This cable also has a barrel connector to provide 5V or 12V power to the SLCD5+N controller. For development kits that use 12V input, it is 2.1mm, center pin positive. For development kits with 5V input, it is 2.5mm, center pin positive. This prevents a 12V supply from being used with a 5V kit.
5. The COM1 cable provides the connection to the “AUX” RS232 serial port. It connects to J7 of the SLCD5+N controller. This typically used as the communications path for downloading of bitmaps and macros to the SLCD5 controller.

4.9 Communications Interface

- The default communication is at a baud rate of 115200 with no parity, software (XON/XOFF) flow control, 8 bits of data, and 1 stop bit. The baud rate can be set to a different initial value on power-on by using the [SET BAUD RATE \(Sticky\)](#) command or the [POWER-ON MACRO](#) feature.
- Only one com port can be in control of the SLCD5+N at any time; control may be switched to the other port using software. (See [Switching Between PC and Embedded Controller](#)).
- 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 values start at the upper left-hand corner. This is point x=0, y=0. The lower right corner is point x=(width-1), y=(height-1); for VGA, width=640 and height =480.
- The maximum length of any command including the termination character is 127 characters.

4.10 Input Buffer Processing

Input Buffer

The SLCD5+N has a nominal 512 byte input circular buffer. As commands are received, they are queued in the buffer and executed first come first served. After a command has been processed, the SLCD5+N 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 SLCD5+N will issue a prompt for each command after it executes. These prompts may be issued while the host is sending a command to the SLCD5+N (full duplex operation).

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

The SLCD5+N controller issues a prompt when it has finished processing a command. This includes the null command which is just a <return>.

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

Flow Control

The SLCD5+N implements software flow control using the XON (decimal 17) and XOFF (decimal 19) characters. When the circular buffer is approximately $\frac{3}{4}$ full, an XOFF is issued to the host. An XON is then issued when the buffer is approximately $\frac{1}{4}$ full. If the host cannot or does not want to accommodate software flow control, the host can make sure that no more than two commands are outstanding at any time. Given that the maximum length of any command is 127 bytes, this guarantees that the host will not be sent an XOFF character.

Buffer Limit Discussion

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

Scenario #1: The host sends data that a) does not conform to the command specification, and b) keeps doing so until the buffer size limit is reached, and c) ignores the XOFF request from the SLCD5+N. 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 SLCD5+N. The limit event can occur when the buffer is full of unexecuted commands.

In both of the previous cases, when the SLCD5+N 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 SLCD5+N can issue the following prompts. These are in addition to any result of a command or button press event.

- `\>'<return>` Indicates that the 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.

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.

4.11 Touch Interface

Depending on the hardware version, the SLCD5+N contains a touch controller that interfaces to either a four wire resistive touchscreen, or an I2C PCAP touch screen.

Touch sensitive areas of the display are defined as either “hotspots” or “buttons”. When these are pressed or released, the SLCD5+N can either notify the host directly or execute a “macro”, or both. A macro is a predefined sequence of SLCD5+N commands.

Hotspot

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

Host Notification

When a touch sensitive area is pressed or released, the SLCD5 can either notify the host, execute a macro or both. See the [BUTTON DEFINE](#) and [TOUCH MACRO ASSIGN](#) commands for details.

4.12 Host Input Processing

When integrated into a host environment, the SLCD5+N sends prompts, touch activity notifications, and user-defined text to the host it is connected to. In general, all SLCD5+N messages are terminated with a <return>.

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

4.13 System Firmware Boot Process

The sequence of operations that occur after power is applied to the SLCD5+N is important for system configuring, macro application programming (via bitmaps and macros), demo execution, and updating firmware. These operations are indicated by output activity on the serial connection COM0 and LED (D2). Users can observe text messages of the Boot Process with a terminal emulator running on a PC when connected to the RS232 port on the COM0 cable.

The System Software consists of two major components: the Bootloader and the main Firmware. The Bootloader is responsible for hardware component initialization, and Firmware updating. The Firmware is responsible for the standard board application functionality as outlined in this document. The [Algorithm](#) section below explains the high level sequence of major operations between these major software components. See sections on [System Configuration](#) and [Bitmaps and Macros](#).

Algorithm

The following pseudo code is an explanation of the System Software Boot Process. Note that files are just checked for presence (e.g. FLASH?.INI), but must be non-zero length.

Boot Loader begins execution

Hardware Component Initialization

Set COM0 baud rate to 115200

Display version and copyright

Look in SD Card for “*.ELF” firmware update file

Read last saved firmware file name from EEPROM

IF the two file names do not match (SD card has different firmware file) THEN

 Update firmware in Flash (erase / program)

 Display status

ENDIF

Display “Starting firmware”

Jump to Firmware code

Boot Loader completed

Firmware begins execution

Read EEPROM

IF EEPROM contents not valid THEN

 Set Default EEPROM Values

ENDIF

IF SD Card present THEN

 IF CONFIG.INI file present THEN

 Parse option keywords, set option variables

 IF verbose = 1 THEN

 Display (option keyword and value)

 ENDIF

 ENDIF

 IF SPLASH.BMP file present THEN

 Display SPLASH.BMP

 ENDIF

 IF SLCD*.BIN present THEN

 IF FLASH2.INI file present THEN

 Copy .BIN file to CPU board flash (28MB)

 Use FLASH for working set BMP/macro/fonts

 ELSE

 Copy .BIN file to DRAM, use as working set BMP/macro/fonts

 ENDIF

 ELSE

 IF FLASH memory contains a .BIN file THEN

 Use flash for working set BMP/macro/fonts (no copy for speed)

 ENDIF

 ENDIF

```
IF RUNDEMO.INI present THEN
  IF "demomac = <macro number>" found THEN
    Run <macro number>
    Display message on main port if "verbose = 1" previously found
  ENDIF
ENDIF

ELSE //SD Card NOT Present
  IF FLASH memory contains a .BIN file THEN
    Use FLASH for working set BMP/macro/fonts (no copy for speed)
  ENDIF
ENDIF
```

Start main firmware application

Figure 10: System Software Boot Algorithm

5. Software Command Reference

5.1 Commands Affecting Non-Volatile Settings (EEPROM)

Some commands affect settings that are stored in non-volatile memory; this memory is implemented using an I2C Serial EEPROM with a limit of 1 million write cycles.

Warning: At 1 write per second, 24 hours a day, the EEPROM will exceed 1 million writes and become unreliable in only 11 days. Take care that commands marked as affecting EEPROM non-volatile settings are not continuously executed.

IT IS RECOMMENDED that commands writing to EEPROM be used only in development, and that the config.ini file (Section 3.4) method of setting these values be used in production.

5.2 Command Basics

All commands end with a <return> character, which is hex 0x0D, decimal 13. All responses from the SLCD5+N end with a <return> character. Linefeeds are not provided in order to minimize communications overhead.

Arguments in angle brackets <..> are to be replaced by the characters described in the command arguments. The angle brackets themselves are not to be used. Square brackets with a vertical spacer '|' indicates optional arguments.

All commands that take *x* and *y* position arguments are affected by the SET ORIGIN command. This allows a set of graphic commands to draw the same object in different locations.

Compressed Syntax

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

```
p <pixels>
```

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

```
p2<return>
```

sets the line width to two.

Case Sensitivity

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

Assumed Orientation

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

Touch Priority

If touch areas overlap, the one with the lowest index value (lowest button or hotspot number) has priority.

5.3 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>]	
Animated GIFs	Animated GIF Start	agif <n> <index> <x> <y> <repeat>
	Animated GIF Stop	agifd
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 Image File	xif <filename>.<ext> 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	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	<code>bd <n> x y type "text" dx dy bmp0 bmp1</code>
Clear All Hotspot	<code>xc all</code>
Clear Hotspot	<code>xc <n></code>
Define Hotspot	<code>x <n> x0 y0 x1 y1</code>
Define Relative X Y Hotspot	<code>xxxy <n> x0 y0 x1 y1</code>
Define Relative X-Y Hotspot (Silent – No Beep)	<code>xxynb <n> x0 y0 x1 y1</code>
Define Special Hotspot (Invisible)	<code>xs <n> x0 y0 x1 y1</code>
Define Special Hotspot (Invisible, No Beep)	<code>xsnb <n> x0 y0 x1 y1</code>
Define Special Typematic Touch	<code>xst <n> x0 y0 x1 y1</code>
Define Touch Action ◆	<code>*touchMode[s] [S L W]</code>
Define Touch Calibration	<code>*tcTimeout [<timeout in seconds>]</code>
Timeout ◆	
Define Touch Parameters ◆	<code>*touchParm [<samples>]</code>
Define Touch Signal Orientation ◆	<code>*touchSwap [0 1]</code>
Define Typematic Touch Area	<code>xt <n> x0 y0 x1 y1</code>
Disable Touch	<code>xd <n "all"> [<n last>]</code>
Enable Touch	<code>xe[s] <n "all"> [<n last>]</code>
Reset Touch Calibration ◆	<code>*RT</code>
Set Touch Characteristics	<code>xset <n> [+ -][p r t T x]</code>
Set Touch Debounce ◆	<code>*debounce <delay in ms></code>
Query (Latching) State Button	<code>ssb <n></code>
Set (Latching) State Button	<code>ssb <n> state</code>
Set Typematic Parameters ◆	<code>typematic[s] <delay> <repeat></code>
Touch Calibrate ◆	<code>tc</code>
Touch Simulate Momentary	<code>*tsmm</code>
Touch Simulate Press	<code>*tsmp</code>
Touch Simulate Release	<code>*tsmr</code>
Touch Simulate Typematic	<code>*tsmt</code>
Touchscreen On/Off	<code>touch [on/off]</code>

Charts	Binary Chart Values	<code>bcv n number_of_pen_values_to_follow</code>
	Chart Clear Display	<code>cc n</code>
	Chart Define	<code>cd n x0 y0 x1 y1 t dw bv tv bc <pens></code>
	Chart Define with Bitmap	<code>cdb n x y dw bv tv bitmap <pens></code>
	Chart Redefine Backgrd Bitmap	<code>cdbb n bitmap</code>
	Chart Redefine Backgrd Color	<code>cdbc n color</code>
	Chart Redefine Draw Mode	<code>cddm n <on off></code>
	Chart Redefine Pen	<code>cdp n pen width color</code>
	Chart Redefine Retrace Mode	<code>cdrm n <on off></code>
	Chart Reset Pens	<code>cr n</code>
	Chart Values	<code>cv n pen0_value [pen1_value ..]</code>

Clear / Unclear	Clear All Hotspot	xc all
	Clear Hotspot	xc <n>
	Clear Screen	Z
	Clear Screen Special	zs <bitmap index>
Com Port	Communication Watchdog Timer	*comwdt <macro> <short> <long> (see description)
	Control Port Autoswitch ◆	aout "<text string>"
	Output String (Aux)	out "<text string>"
	Output String (Main)	ain[b]
	Read from Aux Port	*auxEsc <hex value of ASCII character>
	Set Aux Escape ◆	auxIO [0 1]
	Set Aux Port Auto Read ◆	baudN [460800 230400 115200 57600 38400 19200 9600 4800 1200]
	Set Baud Rate ComN (N = 0 or 1)	baudp[0 1] <rate>
	Set Baud Rate (Prompt First)	bauds[0 1] <rate>
	Set Baud Rate (Sticky and Prompt First) ◆	
Set Control Port ◆	*com[0 1]main[s]	
Set Previous Control Port	*prevCons	
Debug / Maint.	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	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>
	External Memory Available	xma
	External Memory Chip Erase	xmc FEEB

Macros	Debug Macro Get Variable Get Variable (HEX) List Macros Detail Macro Abort Macro Execute Macro Notify Memory Pop Memory Push Power-On Macro ◆ Set Variable Set Variable (HEX) Touch Macro Assign Touch Macro Assign Quiet Touch Macro Assign With Parameters Wait	<pre> *macdebug <0 1> get <internal variable name> getx <internal variable name> lsmac [index] *abt m <n> [macro parameters . . .] *macnote <0 1 2 3> mpop [index] <number> mpush [index] "<string>" [max] *PONMAC <index name> [<option>] set <internal variable name> <value> setx <internal var. name> <HEX value> xm <touch index><macro index name> [<macro2 index>] xmq <touch index> <macro index name> [<macro2 index>] xa[q] <n> action <macro index name> <args> w <number of milliseconds> </pre>
Read / Write / Verify	Controller Type Copy Flash To DRAM CRC External Flash CRC Processor Bootload Code CRC Processor Code CRC Screen Display Config String EEPROM Read / Write Get Panel Type List Downloaded Records Read Frame Buffer Line Read Temperature Set LED Version	<pre> *ctype *flash2ram *CEXT [<from> <to>] *CSUMB *CSUM *CRC *config *eer <hex location> *eev <hex location> <hex value> *panel ls *FB <line> temp led [0 1] vers </pre>
Screen Control	Define Inverter Control Polarity ◆ Define Inverter PWM Control ◆ Define Max, Min Brightness ◆ Define Panel Orientation ◆ Display On/Off External Backlight Bright Ctrl ◆	<pre> *invEnaHiTrue [0 1] *pwmIsEnable [0 1] *maxBrite <value from 0 to 255> *minBrite <value from 0 to 255> *orient [0 1 2 3] v <on off> xbb [+ -]<level> </pre>

	External Backlight Bright Ctrl	xlbs [+ -] <level>
	External Backlight On/Off	xbl <on off>
	Panel Timing Adjust ◆	*paneladj
SD Card	Change SD Card Directory	*cd <directory name>
	Last Firmware File Loaded	*firmware
	List SD Card Directory	*ls
	Load System File From SD Card (Current Directory)	*load (or *LOAD)
	Load System File From SD Card (Specified Directory)	*sdload <directory name>
	Load System File From SD Card Into Flash	*loadf (or *LOADF)
	Save Configuration To SD Card	*getConfigAsFile
	Save Screen Shot To SD Card	*getScreenAsBMPFile
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 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]
	Pixel Write	pwx x y [color]
Sliders	Binary Notification Mode	*binr <0 1>
Meters & Levelbars	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>
	Speaker On / Off ◆	*spkr [on off]
Scrolling Textbox	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>
Text Flashing ListText	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?
	Text Flashing Synchronize	tfs
	UTF8 Enable / Disable	utf8 [on off]
Text	Clear Text Window	twc <n>
Window	Copy Text Window Visible to Variables	twv <n> <line v_index> <char v_index>
	Define Text Window	twd <n> <x0> <y0> <x1> <y1>
	Goto Position in Text Window (Absolute)	twga <n> <line> <char>
	Goto Position in Text Window (Percentage)	twgp <n> <line %> <char %>
	Load Text Window Buffer	tw <n> "text"
	Move Text Window	twm <n> <dir> [count]
	Query Text Window	twq <n>
	Redraw Text Window	twr <n>

5.4 Commands Alphabetically by Command Syntax

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

Command

*abt
*auxEsc <hex value of ASCII character>
*binr <0|1>
*cd <directory name>
*CEXT [<from> <to>]
*com[0|1]main[s]
*comwdt <macro> <short> <long>
*CRC
*CSUM
*CSUMB
*ctype
*debounce <delay>
*debug <0|1>
*eer <hex location>
*eew <hex location> <hex value>
*FB <line>
*firmware
*flash2ram
*getConfigAsFile
*getScreenAsBMPFile
*invEnaHiTrue [0|1]
*load (or *LOAD)

*loadf (or *LOADF)

*ls
*macdebug <0|1>
*macnote <0|1|2|3>
*maxBrite <value from 0 to 255>
*minBrite <value from 0 to 255>
*MFGRESET
*orient [0|1|2|3]
*panel
*paneladj
*PONMAC <index> [<option>]
*prevCons

Command Name

[Macro Abort](#)
[Set Aux Escape](#) ♦
[Binary Notification Mode](#)
[Change SD Card Directory](#)
[CRC External Flash](#)
[Set Control Port](#) ♦
[Communication Watchdog Timer](#)
[CRC Screen](#)
[CRC Processor Code](#)
[CRC Processor Bootload Code](#)
[Controller Type](#)
[Set Touch Debounce](#) ♦
[Debug Touch](#)
[EEPROM Read / Write](#)

[Read Frame Buffer Line](#)
[Last Firmware File Loaded](#)
[Copy Flash To DRAM](#)
[Save Configuration To SD Card](#)
[Save Screen Shot To SD Card](#)
[Define Inverter Control Polarity](#) ♦
[Load System File From SD Card \(Current Directory\)](#)
[Load System File From SD Card Into Flash](#)
[List SD Card Directory](#)
[Debug Macro](#)
[Macro Notify](#)
[Define Max, Min Brightness](#) ♦

[Reset Board to Manufactured State](#) ♦
[Define Panel Orientation](#) ♦
[Get Panel Type](#)
[Panel Timing Adjust](#) ♦
[Power-On Macro](#) ♦
[Set Previous Control Port](#)

```

*pwmIsEnable [0|1]
*RESET
*RT
*sdload <directory name>

*speedtest
*spkr [on|off]
*SPL <index>
*tcTimeout [<timeout in seconds>]
*touchMode[s] [S|L|W]
*touchSwap [0|1]
*touchParm [<samples> <span>]
*tsmm
*tsmp
*tsmr
*tsmt
<return> <return> <return> 3 consecutive
<return> characters to the Aux port
a x0 y0 <radius> <start> <end>
agif <n> <index> <x> <y> <repeat>
agifd
ain[b]
al <alarm> <count>
ani <n> <text string>
ani? <n>
anic
anid <n> [yield #]
anie <n>
anis
anix <n>
aout "<text string>"
auxIO [0|1]
baudN [460800|230400|115200|57600|38400|
      19200|9600|4800|1200]
baudp[0|1] <rate>
bauds[0|1] <rate>

bb <number>
bc <n>
bcv n number_of_pen_values_to_follow <bytes>
bd <n> x y type "text" dx dy bmp0 bmp1

```

[Define Inverter PWM Control](#) ♦
[Reset Board / Software](#)
[Reset Touch Calibration](#) ♦
[Load System File From SD Card \(Specified Directory\)](#)
[Speed Test](#)
[Speaker On / Off](#) ♦
[Splash Screen](#) ♦
[Define Touch Calibration Timeout](#) ♦
[Define Touch Action](#) ♦
[Define Touch Signal Orientation](#) ♦
[Define Touch Parameters](#) ♦
[Touch Simulate Momentary](#)
[Touch Simulate Press](#)
[Touch Simulate Release](#)
[Touch Simulate Typematic](#)
[Control Port Autoswitch](#)

[Draw Arc Segment](#)
[Animated GIF Start](#)
[Animated GIF Stop](#)
[Read from Aux Port](#)
[Alarm](#)
[Animation Define](#)
[Animation List](#)
[Animation Clear](#)
[Animation Disable](#)
[Animation Enable](#)
[Animation Synch](#)
[Animation Delete](#)
[Output String \(Aux\)](#)
[Set Aux Port Auto Read](#) ♦
[Set Baud Rate Port N](#)

[Set Baud Rate \(Prompt First\)](#)
[Set Baud Rate \(Sticky and Prompt First\)](#) ♦
[Beep Touch](#)
[Button Clear](#)
[Binary Chart Values](#)
[Button Define – Momentary](#)

```

bd <n> x y type "text0" "text1" dx0 dy0 dx1
dy1 bmp0 bmp1
bdc <n> x y type "text0" ["text1"] bmp0 bmp1
bdld <index> <offset> <size> <timeout>
beep <count>
beepw <count>
bf[s] [<hertz>]
bv[s] [+|-]<level>
c x0 y0 r [f]
cc n
cd n x0 y0 x1 y1 t dw bv tv bc <pens>
cdb n x y dw bv tv bitmap <pens>
cddb n bitmap
cdbc n color
cddm n <on|off>
cdp <n> <pen> <width> <color>
cdrm n <on|off>
cr n
cv n pen0_value [pen1_value ..]
d [n|x]
e x y <x radius> <y radius>
ef x y <x radius> <y radius>
f <type>
get <internal variable name>
getx <internal variable name>
k x0 y0 x1 y1 <numlines>[l|r|u|d|L|R]
l x0 y0 x1 y1
ld n x0 y0 x1 y1 or inv bv bc <levels>
led [0|1]
ls
lsbmp [index]
lsmac [index]
lv n val
m <n> [macro parameters . . ]
md <id> <bitmap> <x> <y> <type> <minVal>
<maxVal> <initial_value> <minAngle>
<maxAngle> <x0, y0 ... [x10, y10]>
mdb <ix> <bm> <x> <y> <type> <minVal>
<maxVal> <init_val> <minAngle> <maxAngle>
<bandMin> <bandMax> <bandRadius> <bandPen >
<bandColor> <bandBG> <pivotX> <pivotY> <x1 y1
. . .[x10, y10]>

```

[Button Define – Latching State](#)

[Button Define Center Text](#)

[Binary Download](#)

[Beep Once](#)

[Beep Wait](#)

[Beep Frequency](#)◆

[Beep Volume](#)◆

[Draw Circle](#)

[Chart Clear Display](#)

[Chart Define](#)

[Chart Define with Bitmap](#)

[Chart Redefine Background Bitmap](#)

[Chart Redefine Background Color](#)

[Chart Redefine Draw Mode](#)

[Chart Redefine Pen](#)

[Chart Redefine Retrace Mode](#)

[Chart Reset Pens](#)

[Chart Values](#)

[Set Draw Mode](#)

[Draw Ellipse](#)

[Draw Filled Ellipse](#)

[Set Font](#)

[Get Variable](#)

[Get Variable \(HEX\)](#)

[Scroll Screen Area](#)

[Draw Line](#)

[Levelbar Define](#)

[Set LED](#)

[List Downloaded Records](#)

[List Bitmaps Detail](#)

[List Macros Detail](#)

[Levelbar Value](#)

[Macro Execute](#)

[Meter Define](#)

[Meter Define Band](#)


```

mpop [index] <number>
mpush [index] "<string>" [max]
mv <id> <value>
mvb <ix> <val> <bandMin> <bandMax>
<bandRadius> <bandPen> <bandColor>
<indicatorColor>
o <x> <y>
out "<text string>"
p <pixels>
pf x0 y0 [x/y x/y ...]
pfr <angle> x0 y0 [x/y x/y...]
pg x0 y0 [x/y x/y...]
pgr <angle> x0 y0 [x/y x/y...]
pl x0 y0 [x/y x/y...]
plr x0 y0 [x/y x/y...]
ppgr <angle> x0 y0
pr
pw x y [color]
pwx x y [color]
r x0 y0 x1 y1 [style] [color]
rb <on> <off> [alarm]
s <fore> <back>
S <fore_detail> <back_detail>
sc x y
set <internal variable name> <value>
setx <internal variable name> <HEX value>
sl idx bg x y slider off ornt inv cont hi lo
sr
ss
ssb <n>
ssb <n> state
sta <index> "text"
stc <index>
std <index> <type> <x0> <y0> <x1> <y1>
stp <index> "text"
std <index>
sv idx val
t "text string" x y [R|T|X|TR|N]
t "text string"
t? "text string"
ta [L|C|R][T|C|B]
tc

```

[Memory Pop](#)
[Memory Push](#)
[Meter Value](#)
[Meter Value Band](#)

[Set Origin](#)
[Output String \(Main\)](#)
[Set Pen Width](#)
[Draw Filled Polygon](#)
[Draw Rotated Filled Polygon](#)
[Draw Outline Polygon](#)
[Draw Rotated Polygon](#)
[Draw Polyline](#)
[Draw Rotated Polyline](#)
[Redraw Rotated Polygon](#)
[Pixel Read](#)
[Pixel Write](#)
[Pixel Write](#)
[Draw Rectangle](#)
[Beep Repeat](#)
[Set Color \(Basic\)](#)
[Set Color \(Detailed\)](#)
[Set Cursor](#)
[Set Variable](#)
[Set Variable \(HEX\)](#)
[Slider Define](#)
[Restore Drawing Environment](#)
[Save Drawing Environment](#)
[Query \(Latching\) State Button](#)
[Set \(Latching\) State Button](#)
[Append to Scrolling Textbox](#)
[Clear Scrolling Textbox](#)
[Define Scrolling Textbox](#)
[Prepend to Scrolling Textbox](#)
[Query Scrolling Textbox](#)
[Slider Value](#)
[Text Display](#)

[Get Text Display Width In Pixels](#)
[Set Text Alignment](#)
[Touch Calibrate◆](#)

temp
tf <index> [t] "text string" x y [R|T|X|TR]
tf?
tfc
tfd <index> <state>
tfe <index>
tfs
tfx <index>
tm [R|T|X|TR|N]
touch [on/off]
tr x0 y0 x1 y1 x2 y2 [color]
tw <n> "text"
twc <n>
twd <n> <x0> <y0> <x1> <y1>
twga <n> <line> <char>

twgp <n> <line %> <char %>

twm <n> <dir> [count]
twq <n>
twr <n>
twv <n> <line v_index> <char v_index>

typematic[s] <delay> <repeat>
utf8 [on|off]
v <on|off>
vers
w <number of milliseconds>
wr x y [index]
wrf <x> <y>
wrr x y <width> <height> <index> [<address>]
ws x0 y0 x1 y1 [index]
wvr [<line>]
x <n> x0 y0 x1 y1
xa[q] <n> action <args>
xbb[s] [+|-]<level>
xbl <on|off>
xc <n>
xc all
xd <n | "all"> [<n last>]
xe[s] <n | "all"> [<n last>]
xi <index> x y

[Read Temperature](#)
[Text Flashing Display](#)
[Text Flashing List](#)
[Text Flashing Clear](#)
[Text Flashing Disable](#)
[Text Flashing Enable](#)
[Text Flashing Synchronize](#)
[Text Flashing Delete](#)
[Set Text Mode](#)
[Touchscreen On/Off](#)
[Draw Triangle](#)
[Load Text Window Buffer](#)
[Clear Text Window](#)
[Define Text Window](#)
[Goto Position in Text Window \(Absolute\)](#)
[Goto Position in Text Window \(Percentage\)](#)
[Move Text Window](#)
[Query Text Window](#)
[Redraw Text Window](#)
[Copy Text Window Visible to Variables](#)
[Set Typematic Parameters](#) ♦
[UTF8 Enable / Disable](#)
[Display On/Off](#)
[Version](#)
[Wait](#)
[Window Restore](#)
[Wait for Refresh](#)
[Window Restore Rectangle](#)
[Window Save](#)
[Wait Vertical Retrace](#)
[Define Hotspot](#)
[Touch Macro Assign w/ Parameters](#)
[External Backlight Brightness](#) ♦
[External Backlight On/Off](#)
[Clear Hotspot](#)
[Clear All Hotspot](#)
[Disable Touch](#)
[Enable Touch](#)
[Display Bitmap Image](#)

```

xic <index> x y x0 y0 x1 y1
xif <filename>.<ext> x y
xim <index> x y
xio <bitmap index> <x> <y> <0|1> <length>
<offset>
xm <touch index><macro index | name> [<macro2
index | name>]
xma
xmc feeb
xmq <touch index><macro index> [<macro2 index>]
xs <n> x0 y0 x1 y1
xset <n> [+|-][p|r|t|T|x]
xsnb <n> x0 y0 x1 y1

xst <n> x0 y0 x1 y1
xt <n> x0 y0 x1 y1
xxy <n> x0 y0 x1 y1
xxynb <n> x0 y0 x1 y1

y <milliseconds> | stop
z
zs <bitmap index>

```

[Display Clipped Bitmap Image](#)
[Display Image File](#)
[Display Bitmap Image Centered](#)
[Display Windowed Bitmap Image](#)

[Touch Macro Assign](#)

[External Memory Available](#)
[External Memory Chip Erase](#)
[Touch Macro Assign Quiet](#)
[Define Special Hotspot \(Invisible\)](#)
[Set Touch Characteristics](#)
[Define Special Hotspot \(Invisible, No Beep\)](#)
[Define Special Typematic Touch](#)
[Define Typematic Touch Area](#)
[Define Relative X-Y Hotspot](#)
[Define Relative X-Y Hotspot \(Silent No Beep\)](#)
[Animation Yield](#)
[Clear Screen](#)
[Clear Screen Special](#)

5.5 COMMANDS

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.

ANIMATED GIF START

Description: Starts an animated GIF. Note that the GIF file should be located in RAM to provide the best performance. Use the [COPY FLASH to DRAM](#) command before starting an animated GIF (only required once per reset/power cycle). Note: Only one animated GIF may be running at a time.

Command: `agif <n> <index> <x> <y> <repeat>`

Arguments: `<n>` - Animated GIF number.
`<index>` - Index of GIF (in bitmap list) to display.
`<x> <y>` - Top/left corner of GIF screen location.
`<repeat>` - Number of times to cycle GIF. Use 0 to repeat forever.

Example: `agif 0 1 0 0 8`
Starts animated GIF 0 using GIF image 1 at coordinates 0/0, repeating eight times.

ANIMATED GIF STOP

Description: Stop animated GIF.

Command: `agifd`

Arguments: None.

Example: `agifd`

ANIMATION CLEAR (TEXT FLASHING CLEAR)

Description:	Clears the animation and flashing text definitions and disables the animation engine.
Command:	<code>anic</code> or <code>tfc</code>
Arguments:	None.
Example:	<code>anic</code> Clears the animation buffers and stops the animation engine.

ANIMATION DEFINE

Description:	<p>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).</p> <p>Animations are disabled when defined and must be activated using the “<code>anie</code>” (Animation 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 “<code>anid</code>” command is issued to stop the animation.</p> <p>An animation executes in a temporary state (see SAVE DRAWING ENVIRONMENT), which exists only until the animation yields. Properties like foreground color must be set within the yield point where they are to be used.</p>
--------------	--

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, 0-9.
`<Yield #>` A reference to one of a animation's yield commands.

Yields are numbered for each animation starting at 0 (zero) and continues up to the number of yields-1 contained in that animation.

Note: Animations are "stopped" by advancing and executing those commands between the previous and selected yield.

Example: `anid 0 0`
Stops animation 0 at the first yield command.

ANIMATION ENABLE

Description: Enables animation execution for a specified animation.

Command: `anie <n>`

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

Example: `anie 0`
Enables animation 0.

ANIMATION LIST (TEXT FLASHING LIST)

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

Command: `ani? <n> 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, after issuing ``f12``, ``s 0 1``, and ``z`` commands:

```
f12
>
s 0 1
>
z
>
tf 0 "hello world"
>
ani? 0 Returns:

// ani:0 count: 500 Size: 134
```



```

ani 0 f 12
ani 0 S 000000 fffffff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm N
ani 0 T "hello world"
ani 0 y 500
ani 0 f 12
ani 0 S fffffff fffffff
ani 0 ta LT
ani 0 o 0 0
ani 0 sc 0 0
ani 0 tm N
ani 0 T "hello world"
ani 0 y 500
>

```

ANIMATION SYNCH

Description: Restarts all animations at beginning of scripts.

Command: `anis`

Arguments: None.

Example: `anis`

Any running animations are restarted.

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.

ANIMATION YIELD

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

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

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

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

Stop Halt this animation until the ANIMATION ENABLE command is issued.

APPEND TO SCROLLING TEXTBOX

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:	<code>sta <index> "text"</code>
Arguments:	<code><index></code> Index of scrolling textbox. <code>"text"</code> Text to append, in double-quotes.
Example:	<code>sta 0 "Sample text"</code>

BEEP FREQUENCY, BEEP FREQUENCY SPECIAL

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

Description:	Sets the frequency of the beeper; 'bf' stores the setting in non-volatile memory (see warning), whereas 'bfs' does not. When given no arguments, the command returns the current setting. 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:	<code>bf[s] [<hertz>]</code>
Arguments:	<code><hertz></code> is number from 1 through 4000. Default is 2650, but may be slightly different due to volume calibration at the factory. If no argument is supplied, the current frequency is returned as a variable length decimal number.
Example:	<code>bfs 2500</code> Sets the beep frequency to 2500 Hertz, but does not store the setting; on powerup, the value stored in non-volatile memory will be used. <code>bf (or bfs)</code> Returns 2500 after the above command was issued.

BEEP ONCE

Description:	Beeps the beeper for <code><count></code> ms. This will temporarily interrupt any running repeating beep. A prompt is returned immediately even if the beep continues.
Command:	<code>beep <count></code>
Arguments:	<code><count></code> is number of ms to sound the beeper. The value has the range of a 32-bit variable.

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 value has the range of a 32-bit variable.

<off> is number of ms to stay silent before beeping again. The value has the range of a 32-bit variable.

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

Example: rb 100 400

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

BEEP TOUCH

Description: Sets the duration of the audible feedback beep when a hotspot or button is pressed. Not stored in non-volatile memory. Default is 10 which equals 100ms beep.

Command: bb <number>

Arguments: <number> is tens of milliseconds to sound the beeper.

Example: bb 10

Sets the beep feedback to power-on value.

BEEP VOLUME, BEEP VOLUME SPECIAL

Description: Sets the volume level of the beeper; ‘bv’ stores the setting in non-volatile memory ([see warning](#)), whereas ‘bvs’ does not. When given no arguments, the commands return the current setting.

Command: bv[s] [+|-]<level>

Arguments: <level> is number from 0 through 255. Default is 200. Loudest is 255. The optional ‘+’ or ‘-’ prefix changes the <level> into an increment up or down.

Example: bvs 120

Sets the beep volume setting to 120, but does not store the setting; on powerup, the value stored in non-volatile memory will be used.

bv (or bvs)

Returns 120 after the above command was issued.

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 value has the range of a 32-bit variable.

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 five 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.
- `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 values of 0,5,10,15,20 for the teal pen and 0,10,20,30,40 for the red pen. Each data point is four pixels to the right of the last one. Note: 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 SLCD5+ 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 SLCD5 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 SLCD5x 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 SLCD5 Flash memory must be erased before bdld can be used to update its contents. The erase command is “xmc FEEB”.

BINARY NOTIFICATION MODE

Description: Used to set SLCD5 notification mode to binary or ASCII.
Due to parsing constraints, it is sometimes useful to have the SLCD5 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.
2 Button / Hotspot / Macro notification is in binary format as follows:

Functionality:

Standard (ASCII) Notification	Binary Notification
<code>x<index><return></code>	<code>X<binary index></code>
<code>x<index> <Xr> <Yr><return></code>	<code>Y<index_Byte><Xr_LSByte><Xr_MSB yte> <Yr_LSByte><Yr_MSBByte></code>
<code>r<index><return></code>	<code>R<binary index></code>
<code>s<index> <state><return></code>	<code>S<binary index><binary state></code>
<code>m<index><return></code>	<code>M<binary index></code>
<code>e<index><return></code>	<code>E<binary index></code>
<code><index></code> is 1-3 ASCII digits	<code><binary index></code> is a single 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.

Note: To clear all buttons, see the [CLEAR ALL HOTSPOT](#) "xc all" command.

BUTTON DEFINE CENTER TEXT

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

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

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

`x y` Upper left hand corner of the button bitmap.

`type` Button type:

All types supported in `BUTTON DEFINE – LATCHING STATE` and `BUTTON DEFINE – MOMENTARY` commands.

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

`"text1"` 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.

`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" 2 3`

Defines button number 23 displayed at `x=150, y=100`. The “un-pressed” image uses bitmap 2 with the text “Test” drawn on the bitmap in the vertical and horizontal center of the bitmap. The “pressed” image is the same except bitmap 3 is used.

Host notification: See [BUTTON DEFINE – MOMENTARY](#) command.

Example 2: `bdc 24 150 200 2 "ON" "off" 2 3`

Defines button number 24 displayed at `x=150, y=200`. The “un-pressed” image uses bitmap 2 with the text “ON” centered on it. The “pressed” image uses bitmap 3 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 it 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.
 - 20 Latching. Same as above. Initial state is set to state 0.
 - 21 Latching. Same as above, but with initial state set to state 1.
- `"text0"` Text string to be displayed on the button in state 0. The current foreground color will be used for the text. For multi-line text, use the newline (`'\n'`) character decimal 10.
- `"text1"` Text string to be displayed on the button in state 1. The current foreground color will be used for the text.
- `<dx0>` Text offset in the x direction from the upper left-hand corner of the button for `"text0"`.
- `<dy0>` Text offset in the y direction from the upper left-hand corner of the button for `"text0"`.
- `<dx1>` Same as above for `"text1"`.
- `<dy1>` Same as above for `"text1"`.
- `<bmp0>` Index of bitmap displayed in state 0.
- `<bmp1>` Index of bitmap displayed in the state 1.

Note: Both bitmaps must be the same size.

Host notification: `s<n> <s><return>` where `<s>` is 0 or 1 for the new state. Note the space between the button index and the state value.

Example:

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

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

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

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

BUTTON DEFINE – MOMENTARY

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

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

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

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

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

`<type>` Button type:

- 1 Standard. Displays `<bmp0>` normally, and `<bmp1>` when pressed. Host is notified when button is pressed, but not when it is released.
- 3 Typematic. Same as regular but with typematic functionality; that is, host notification repeats after the button is held down. See [SET TYPEMATIC PARAMETERS](#) command.
- 30 Typematic; same as type 3 above, except that subsequent host notifications do not generate a beep.
- 4 Standard except host is notified only when the button is released.
- 5 Standard with both press and release notification.

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

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

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

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

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

Note: Both bitmaps must be the same size.

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

`x<n><return>`

Host notification, type 4, 5 when button released:

`r<n><return>`

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 bitmap 6 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>`

CHANGE SD CARD DIRECTORY

Description: Changes the current working directory of the installed SD card. This can be used in conjunction with the [xif](#) and [*LOAD](#) commands.

Command: `*cd <directory name>`

Note: <directory name> follows the DOS 8.3 naming convention, so the directory name is limited to eight characters.

Example: `*cd /Images`

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 [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)*

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 in RGB444 format (see [Color Specifications](#)).

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 [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) , positive numbers only.

`tv` – Top data value (highest `y` value) , positive numbers only.

`bitmap` – Bitmap index

`<pens>` - one or more sets of two values: pen width and pen color; up to eight 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 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.

CLEAR ALL HOTSPOT

Description: Clears all previously defined 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 (touch button) number. Value must be in the range of 128 to 255.

CLEAR SCREEN

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

Command: `z`

CLEAR SCROLLING TEXTBOX

Description: Clears an existing Scrolling Textbox.

Command: `stc <index>`

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

Example: `stc 0`

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 TEXT WINDOW

Description: Clear a Textwindow viewport and its associated text buffer. The Textwindow viewport position is reset to the upper-left corner of the associated text buffer (first character position of first line position).

Command: `twc <n>`

Arguments: `<n>` - Textwindow number.

Example: `twc 0`

CONTROL PORT AUTOSWITCH

Description: The SLCD5 unit has two serial ports. COM0 is connected to J6 while COM1 is connected to J7. Only one port is active at a time as the unit's control port. However in certain circumstances it is useful to be able to switch ports temporarily.

Command: This can be done by sending three consecutive special characters to the inactive port. Once this is done, the inactive port will become the active port temporarily.

The special character is `<return>` by default. The `*auxEsc` command can be used to change the special character.

CONTROLLER TYPE

Description: Returns type of controller – SLCD5 or SLCD5+.

Command: `*ctype`

Returns: 0 – SLCD5
1 – SLCD5+ or SLCD5+N
! – Error prompt for older firmware; controller type is SLCD5

COPY FLASH to DRAM

Description: If the flash memory is being used directly as the working set for Bitmaps/macros/fonts (no SD card present), the image load time can be slower than if the working set were in DRAM. The flash is used directly so that boot time from power-on is fast. If the extra speed is needed, this command copies the working set to DRAM. This command may take several seconds to execute depending on the size of the .bin file loaded in flash.

Command: `*flash2ram`

Returns: Standard prompt; error prompt means the flash contents were not valid.

COPY TEXT WINDOW VISIBLE TO VARIABLES

Description: Copy the value 0-100 representing the visible (viewport) line location (by percent) into the system integer variable referenced by the “line v_index” and the visible character location (by percent) into the system integer variable referenced by the “char v_index” line. Line values are 0-100, where 0 means showing bottom of buffer and 100 means top line. Character values are 0-100, where 0 means showing left of buffer and 100 means right.

Command: `twv <n> <line v_index> <char v_index>`

Arguments: `<n>` - Textwindow number.
`<line v_index>` - Integer variable to receive the viewport line position.
`<char v_index>` - Integer variable to receive the viewport char position.

Example: `twv 0 2 3`
Copies Textwindow 0 current viewport line percentage number into integer variable i2 and the current viewport character percentage number into integer variable i3.

CRC EXTERNAL FLASH

- Description: Returns the 16-bit CRC of the data flash used to store macros and bitmaps. This can be used in production code to verify that the correct bitmaps are loaded in the board.
- Command: *CEXT [<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 BOOTLOAD CODE

- Description: Returns a 16-bit CRC of the processor bootload code space (first 64KB of program flash).
- Command: *CSUMB

CRC PROCESSOR CODE

- Description: Returns a 16-bit CRC of the entire processor code space. The purpose is to give an indication of the firmware version loaded.
- Command: *CSUM
- Returns: 0xHHHH<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.
- Returns: 0xHHHH <return> where H is a single hex digit.

DEBUG MACRO

- Description: Used to enable macro debug. When set, the commands in the macro are displayed as they are executed. This is useful when a macro stops due to a command error.
- Command: *macdebug <0|1>
- Returns: [on|off]<return>

DEBUG TOUCH

Description: Used for touch screen debugging. When set to 1, an “X” is written on the screen when a valid touch is detected. Also, additional information is displayed during touch calibration.

Command: `*debug <0|1>`

Returns: `[on|off]<return>`

DEFINE HOTSPOT (VISIBLE TOUCH AREA)

Description: Defines 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 hotspot.

Returns: `x<n><return>`
when the corresponding button is pushed. Note that once a hotspot is defined, the return string can be transmitted at any time including during a command transmission to the unit (full duplex).

Example: `x 135 100 100 179 139`
Creates a rectangular hotspot with width of 80 and height of 40.

DEFINE INVERTER CONTROL POLARITY

Description: The inverter power connector (J13) has an “inverter enable” signal. On some inverters, this is high true (high = typically 5V), and on others it is low true. This command sets the enable polarity and stores the value in non-volatile memory ([see warning](#)); when given no arguments, it returns the current setting.

Command: `*invEnaHiTrue [0|1]`

Argument: (none) - Display current state.
0 - Inverter enable signal will be low for inverter on.
1 - Inverter enable signal will be high for inverter on.

DEFINE INVERTER PWM CONTROL

- Description: The backlight inverter is typically either voltage- or enable-controlled. This corresponds to high or low PWM frequency respectively. If the inverter is voltage-controlled the PWM must be high frequency so the analog filter will work. This command sets the value and stores it in non-volatile memory ([see warning](#)); when given no arguments, it returns the current setting.
- Command: `*pwmIsEnable [0|1]`
- Arguments: (none) - display current state.
0 - pwm is high frequency for voltage controlled inverters.
1 - pwm is low frequency for “enable” controlled inverters.

DEFINE MAX, MIN BRIGHTNESS

- Description: These commands set the range and polarity for inverter brightness control via J13, and store the values in non-volatile memory ([see warning](#)). When given no arguments, they return the current setting.
- Command: `*maxBrite <value from 0 to 255>`
`*minBrite <value from 0 to 255>`
- Arguments: If `maxBrite > minBrite`, the polarity of the brightness signal is positive: higher brightness = higher level.

If `maxBrite < minBrite`, the polarity of the brightness signal is negative: higher brightness = lower level.
- Example: `maxBrite 0`
`minBrite 230`
When the backlight brightness command “`xbb 255`” is issued, the brightness control level will be zero. When “`xbb 0`” is issued, the brightness control level will be $(230/255 * \text{maximum value})$.

DEFINE PANEL ORIENTATION

Description: Some LCD panels have hardware signals to flip the display horizontally, vertically, or both. This command allows these signals to be set and it stores the value in non-volatile memory ([see warning](#)). When given no arguments, it returns the current setting.

Command: `*orient [0|1|2|3]`

Argument: (none) - Display current state
0 - Pin 31 low, pin 30 low (pins on DF9-31 connector)
1 - Pin 31 low, pin 30 high (pins on DF9-31 connector)
2 - Pin 31 high, pin 30 low (pins on DF9-31 connector)
3 - Pin 31 high, pin 30 high (pins on DF9-31 connector)

Example: `*orient 2`

This flips an NEC panel 180 degrees compared to the “`*orient 0`” position.

DEFINE RELATIVE X-Y HOTSPOT

Description: Defines a touch area on the screen that has no reverse video highlight. When touched, this area’s number and the X and Y position of the touch is returned, relative to the top left corner of the area as defined by x0 and y0. A beep is generated.

Command: `xxxy <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 hotspot.

Returns: `x<n> <x> <y><return>`
when the screen is touched in the hotspot area. Note that once a hotspot is defined, the return string can be transmitted at any time including during a command transmission to the unit (full duplex).

Example: `xxxy 140 100 120 200 220`

Enables a rectangular hotspot. When the screen location x=110, y=140 is touched the following string is sent to the host:

```
x140 10 20<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

Description: Define a new Scrolling Textbox, using the current font, foreground color, and background color. Note that scrolling textboxes require monospaced fonts. The built-in fonts that have names starting with ‘M’ or ‘m’ are monospaced fonts.

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

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

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

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 and beeps only once when pressed continuously (use “xset <n> -T” to enable repeated beeps).

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

Arguments,
Returns: Same as DEFINE HOTSPOT command.

DEFINE TEXT WINDOW

Description: Define a new Textwindow, using the current font, foreground color, and background color. Note that Textwindows require monospaced fonts. The built-in fonts that have names starting with 'M' or 'm' are monospaced fonts. The Textwindow viewport and its associated text buffer will be blank until text is loaded into the buffer with the LOAD TEXT WINDOW BUFFER command. The Textwindow viewport position is set to the upper-left corner of the associated text buffer (first character position of first line position). Textwindows allow up to 200 lines of 120 characters each (SLCD5+).

Command: twd <n> <x0> <y0> <x1> <y1>

Arguments: <n> - Textwindow number, 0 or 1 (2 max).
<x0> <y0> - Top/left corner of Textwindow viewport location.
<x1> <y1> - Bottom/right corner of Textwindow viewport location.

Example: twd 0 10 80 165 180

Defines a Textwindow with a viewport at coordinates 10/80 165/180.

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, it returns the current setting. If the optional 's' is present, it does NOT store the value.

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

Arguments: (none) - Display current options.
S - Standard (none of the others).
L - Lockout other touch areas until a touch release (turn off "n-key rollover).
P - Use touch pressure to validate touch (can eliminate multiple touch aliasing).
W - Wandering from the initial press will cause the touch to release.

Example: *touchModes LPW

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

DEFINE TOUCH CALIBRATION TIMEOUT

Description: The touch screen calibrate command has a timeout when waiting for a touch input. This sets the timeout value and stores it in non-volatile memory ([see warning](#)). When given no arguments, it returns the current setting.

Command: `*tcTimeout [<timeout in seconds>]`

Example: `*tcTimeout 5`

This sets the timeout to five seconds.

DEFINE TOUCH PARAMETERS

Description: The touch screen can have different sensitivities defined using this command. These values are dependent on the touch panel used. The command stores the values in non-volatile memory ([see warning](#)). When given no arguments, it returns the current setting.

Command: `*touchParm [<samples>]`

Arguments:

- (none) - Display current options.
- <samples> - Number of touch samples required for a valid touch; the larger the number the less sensitive.
- - Allowable range for sample location measurement; the smaller the number the less sensitive.

Example: `*touchParm 8 12`

DEFINE TOUCH SIGNAL ORIENTATION

Description: The touch panel connectors J3, J11, J12 are defined in terms of X/left/right and Y/up/down. A touch panel that has these X and Y reversed (swapped) can be accommodated by using this command, which stores the value in non-volatile memory ([see warning](#)). When given no arguments, it returns the current setting.

Command: `*touchSwap [0|1]`

Arguments:

- (none) - Display current state.
- 0 - Touch signal names are given per J3 / J11 / J12 descriptions.
- 1 - Touch signals on J3 / J11 / J12 are swapped X and Y.

DEFINE TYPEMATIC TOUCH AREA

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

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

Arguments,

Returns: Same as DEFINE HOTSPOT command.

DISABLE TOUCH (Hotspot / Button)

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 area or 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 areas 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 fourth 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 fourth 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* (i.e.: x0 y0 x1 y1 are offsets from x y).

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 CONFIG STRING

Description: The SLCD5 configuration can be set by a CONFIG.INI file. This file can also define the response of this command. In this way, the host software can verify the SLCD5 configuration.

Command: *config

Returns: Text string that was set by the line:

config = "text string"

in the [CONFIG.INI](#) file.

Note: The text string displayed by this command does not have double quotes around it.

DISPLAY IMAGE FILE

Description: Displays an image file from the current directory on the SD Card (see [CHANGE SD CARD DIRECTORY](#)) onto the screen at x y (relative to top left corner of the screen, which is location 0, 0).

Command: `xif <filename>.<ext> x y`

Arguments: `<filename>` is the name of the image file (max 8 chars).

`<ext>` is a 3 character extension identifying the type of image file:

“bmp” bitmap

“gif” GIF

“jpg” JPEG

Example: `xif button.bmp 10 20`

This displays the bitmap image in file “button.bmp”, found in the current directory of the SD Card, at location (10, 20).

DISPLAY ON/OFF

Description: Turns power to the display (and backlight) on or off.

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: 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 current 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 current 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 orientated.

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 Orig> <Y Orig>` is X/Y location 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 0 639 479`
This will draw a line from the upper left-hand corner of the screen to the lower right hand corner of a VGA display (640x480).

DRAW OUTLINE POLYGON

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

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

Arguments: `<X Orig> <Y Orig>` is X/Y location 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 POLYLINE

Description: Draws a polyline at the specified origin, using 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 Orig> <Y Orig>` is X/Y location for polygon.
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `pl 100 100 35 -15 0 0 35 15`

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 Orig> <Y Orig> is X/Y location 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 a filled polygon rotated 45 Deg **CCW** 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 Orig> <Y Orig>` is X/Y location 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 CW 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 Orig> <Y Orig>` is X/Y location for polygon.
`<X/Y vertices ...>` are vertex end-points (MAX=11).

Example: `plr 45 100 100 35 -15 0 0 35 15`
Draws a polyline rotated 45 Deg **CCW** 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 argument is supplied, the triangle is also filled with the specified color. Note: To fill without an outline border, set the pen width to one.

Command: `tr x0 y0 x1 y1 x2 y2 [RGB444 | RGB888]`

Arguments: The three x, y sets are the triangle vertices. The optional color fill argument is three or 6 hex characters (see [Color Specifications](#)).

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 OCC`
Same as above, but the triangle is filled with light blue.

EEPROM READ, WRITE

| | |
|--------------|--|
| Description: | The EEPROM can be read from and written to, at addresses 0x00-0x0F (16 bytes total). When writing values, be aware the EEPROM chip has limited total write cycles (see warning). |
| Commands: | *eer <hex address byte>
*eew <hex address byte> <hex value byte> |
| Arguments: | all arguments are ASCII Hex characters, representing byte values. |
| Examples: | *eew 0f a5
*eer 0f |
| Returns: | 0F = A5 |

ENABLE TOUCH (Hotspot / Button)

| | |
|--------------|---|
| 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 area or 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 areas or buttons numbered 2-10. |

EXTERNAL BACKLIGHT BRIGHTNESS CONTROL

Description: Sets the brightness of the external backlight if the external unit supports this feature; 'xbb' stores the setting in non-volatile memory ([see warning](#)), whereas 'xbbs' does not. When given no arguments, the command returns current setting.

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

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

Example: `xbb -10`

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

Example: `xbbs 255`

This set the brightness to maximum and no save in EEPROM (executes quickly).

EXTERNAL BACKLIGHT ON/OFF

Description: Turns the external backlight control on or off via J13.

Command: `xb1 <on|off>`

EXTERNAL MEMORY AVAILABLE

Description: Returns four values: the size of External (macro/bitmap) memory, in bytes, the size of a single sector, the size a second time and the base address of FLASH.

Command: `xma`

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 used to create the firmware. Note: if the CONFIG.INI file specifies a config string, this string will be returned by the *panel command.

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: `getx i9`
A5

Internal Integer variable i9 returns its value, HEX A5.

GOTO POSITION IN TEXT WINDOW (Absolute)

Description: Move the viewport of a Textwindow to a new absolute position of the text buffer. Movement in the 'down' direction will be limited by the actual number of lines of text in the text buffer.

Command: `twga <n> <line> <char>`

Arguments: `<n>` - Textwindow number.
`<line>` - New line position. Use 999 to indicate current position.
`<char>` - New character position. Use 999 to indicate current position.

Example: `twga 0 5 20`
Moves Textwindow 0 viewport to line 5, character 20.
`twga 1 999 75`
Moves Textwindow 1 viewport to character 75 without changing the line position.
`twga 1 150 999`
Moves Textwindow 1 viewport to line 150 without changing the current character position.

GOTO POSITION IN TEXT WINDOW (Percentage)

Description: Move the viewport of a Textwindow to a new position, by percent, of the text buffer. Movement in the 'down' direction will be limited by the actual number of lines of text in the text buffer.

Command: `twgp <n> <line %> <char %>`

Arguments: `<n>` - Textwindow number.

`<line>` - New line position by %. Use 999 to indicate current position.

`<char>` - New character position by %. 999 indicates current position.

Example: `twgp 0 5 20`

Moves Textwindow 0 viewport to the line position 5% into the buffer, character position 20% into the buffer.

`twgp 1 999 75`

Moves Textwindow 1 viewport to the character position 75% into the buffer without changing the line position.

`twgp 1 80 999`

Moves Textwindow 1 viewport to the line position 80% into the buffer without changing the character position.

LAST FIRMWARE FILE LOADED

Description: When the bootloader finds a new firmware file on the SD card, it stores the name of the file in EEPROM. This command displays the name of the most recent firmware file loaded. Fresh from the factory, or after using the [*MFGRESET](#) command, the response will be "????????".

Command: `*firmware`

Returns: "`<eight ASCII characters>`"

This is the eight character DOS filename. The firmware file is this name with extension ".elf".

LEVELBAR DEFINE

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

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

Arguments: `n` – Object index from 0 to 9 (maximum 10 charts).

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

`or` – Orientation: 0 = vertical, 1 = horizontal

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

`bv` – Bottom data value; should be 1 if value 0 means no level displayed; negative numbers allowed

`bc` – Background color in RGB444 or RGB888 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 three sets are possible. Color is the same format as 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 used to display a single bitmap.

LIST DOWNLOADED RECORDS

- Description: Returns a summary of the records in data flash memory. 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 used to display a single macro.

LIST SD CARD DIRECTORY

- Description: Lists the contents of the current working directory of the installed SD card. Directories are listed with the “/” character following their name.
- Command: `*ls`
- Note: Listed files and directories will follow the DOS 8.3 naming convention.
- Example: `*ls`

```
RUNDEMO.INI
RT2_DEMO/
IKB_DEMO/
GFX_DEMO/
SLCD5PWV.BIN
SPLASH.BMP
```

LOAD SYSTEM FILE FROM SD CARD (CURRENT DIRECTORY)

Description: Causes the file “SLCD????.BIN” to be loaded from the SD card current directory into memory; if the optional “SPLASH.BMP” file is present, the image in it will be displayed. For a complete description refer to the Firmware portion of [System Firmware Boot Process](#). Note: the file “SLCD????.BIN” is generated by the BMPload.exe program and contains bitmaps, macros, and external fonts.

Command: *load (or *LOAD)

LOAD SYSTEM FILE FROM SD CARD (SPECIFIED DIRECTORY)

Description: Causes only the bitmap file “SLCD????.BIN” to be loaded from the SD card directory into memory. This file is generated by the BMPload.exe program and contains both bitmaps and macros.

Command: *sdload <directory name> [index]

Arguments: <directory name> – Directory to load .BIN file from.

Note: <directory name> follows the DOS 8.3 naming convention. The <directory name> argument limit is 16 characters so that directories can be specified using the .. and / (Linux)conventions.

[index] – Optional index of macro to run after loading .BIN file.

Example: *sdload /French 1

The file “SLCD????.BIN” (if present) is loaded into memory from the directory “/French” on the SD Card. Macro 1 will be automatically run after the .BIN file is loaded.

LOAD SYSTEM FILE FROM SD CARD INTO FLASH

Description: Same as the *load command except that instead of loading into memory, the file is loaded into flash, and used from there. This can take some time to execute. Status is provided on the board LEDs.

Command: *loadf or *LOADF

Returns: Status messages as progress, standard or error prompt if the command succeeds or fails.

LOAD TEXT WINDOW BUFFER

- Description: Loads a line of text into the buffer for the specified Textwindow. The new text is added to the next available line in the text buffer, until the buffer is full. If the new line of text is within the viewport, the new line is displayed. The viewport will scroll up as long as the bottom of the viewport is set to the bottom of the text buffer.
- Command: `tw <n> "text"`
- Arguments: `<n>` –Textwindow number.
`"text"` – Text in double-quotes.
- Example: `tw 0 "A line of text."`
- Returns the standard reply prompts for success or error (e.g. if text buffer is full).

MACRO ABORT

- Description: This command stops execution of the current running macro. In addition, the command flushes (resets) the incoming command buffer. It is useful for instance if the host has sent a long sequence of commands to update data on a screen. While those commands are being processed by the SLCD5, a button is pushed which means "draw different screen". The host then sends a MACRO ABORT command and waits for the response. The SLCD5 can start drawing the new screen immediately, instead of having to wait until the previously buffered commands were finished.
- 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).

MACRO EXECUTE

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

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

Note: The maximum number of arguments is 10, and the maximum size of arguments is only limited by total command line length (max 128). See [Appendix E](#).

Command: m <n|name> [macro parameters . . .]

Arguments: <n|name> is the macro index number (1 to 254), or the macro name. If the macro takes arguments, the values are supplied in order after the macro number. They are delimited by spaces. If a space is to be included in an argument, the argument must be enclosed with double quotes.

Example: m2

This causes macro #2 to execute.

Example: m test3 “ “ 2

This causes macro “test3” to execute with a value for the first parameter of a space character, and the value of the second parameter the number 2.

Command: m <n|name>:<label> [macro parameters . . .]

Arguments: Same as above, with macro label. See [Appendix E](#) for a full description.

Example: m test3:lbl_4 arg1 arg2

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: 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 . . . – Polygon points for indicator relative to pivot point. Max 10 points.
- [x10 y10]

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.

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 the indicator for a specified meter. The meter must have been previously created by the Meter Define command.

Command: `mv id value`

Arguments: `id` – Index value for meter previously defined.
`value` – Value to set indicator. 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.

MOVE TEXT WINDOW

Description: Move the viewport of a Textwindow to a new position of the text buffer. The viewport may be moved up, down, left or right. Movements may be one or more lines or characters, or to the furthest position in the specified direction. Movement in the 'down' direction will be limited by the actual number of lines of text in the text buffer.

Command: twm <n> <dir> [count]

Argument: <n> - Textwindow number.

<dir> - Direction to move viewport. Options are:

- u - Move viewport up <count> lines.
- d - Move viewport down <count> lines.
- l - Move viewport left <count> characters.
- r - Move viewport right <count> characters.
- U - Move viewport to the top line in the buffer.
- D - Move viewport to the bottom line in the buffer.
- L - Move viewport to the left-most side of the buffer.
- R - Move viewport to the right-most side of the buffer.

[count] - Number of lines or characters to move, depending on <dir>. The default, if not specified, is one line or character.

Host Notification: If the Textwindow viewport is commanded to move beyond the top or bottom limit of the associated text buffer, the position of the Textwindow viewport will be set to the top/bottom line and the host will receive a notification indicating which direction has hit a limit. There are no host notifications for left/right movement limits.

Trying to move Textwindow 0 viewport up beyond the top line of the buffer:

```
u0<return>
```

Trying to move Textwindow 1 viewport down beyond the bottom line of the buffer:

```
d1<return>
```

Examples:

```
twm 0 d 1
```

Moves Textwindow 0 viewport down one line.

```
twm 1 U
```

Moves Textwindow 1 viewport up to top of text buffer.

```
twm 1 r
```

Moves Textwindow 1 viewport right one character position.

OUTPUT STRING (AUX)

- Description:** Similar to OUTPUT STRING, but writes to the AUX communications port. Note that whatever port is acting as the main port cannot be written to this way; use the OUTPUT STRING command. Standard hex escapes are supported. A null byte must be sent as “\x00”.
- Command:** `aout "<your message>"`
- Argument:** Quoted string to send “<your message>”
- Example:** `aout "hello world\x01\xff"`
Sends “hello world” followed by two bytes hex 01 and hex FF to the Auxiliary serial port.
- Example:** `aout "\x00"`
Sends a single byte 0x00 to the Auxiliary port.

OUTPUT STRING (MAIN)

- Description:** This outputs a text string to the serial port. This is typically used in macros that are assigned to buttons using the quiet feature above. This enables a button press to output arbitrary text to the serial port.
- Command:** `out "<text string>"`
- Arguments:** The text string can contain back tick enclosed variables or the following escapes:
- `\\` = `\`
 - `\"` = `"`
 - `\`` = ```
 - `\n` = line feed
 - `\r` = return
 - `\xhh` = arbitrary character with hex value hh
- Example:** `out "\x48ello \"world\"\\r"`
This will send the following string out on the serial port:
Hello "world"<return>

PANEL TIMING ADJUST

- Description:** The SLCD5 firmware sets its LCD panel timings to certain default values. These may not work correctly with certain panels. This command can be used to adjust the timings.
- Command:** `*paneladj`
- Use:** Follow the directions displayed on the terminal emulator. Once the proper timing has been determined, these values can be put in the [CONFIG.INI](#) file for production setting, or stored in non-volatile memory ([see warning](#)).

PIXEL READ

Description: The “pr” commands read a pixel at location (x, y).

Command: `pr x y`

Returns: Color value in RGB565 format (see [Color Specifications](#))

Command: `prx x y`

Returns: Color value in RGB888 format (see [Color Specifications](#))

Example: `prx 10 20` (assume the pw command listed below has been sent)

Returns: `F80008`

PIXEL WRITE

Description: The “pw” commands write a pixel at location (x, y). See [Color Specifications](#) for color definitions.

Commands: `pw x y [RGB565 color value]`
`pw x y [RGB888 color value]`

Arguments: If no color value is given, the pixel is written in the current foreground color.

Example: `pw 10 20 F801`

This sets the pixel at (10, 20) to R=11111XXX (8 bit), G = 0, B = 00001XX.

POWER-ON MACRO

- Description:** Specifies a macro to be executed when the unit is first powered on; it stores the setting in non-volatile memory ([see warning](#)). When given no arguments, the command returns the current setting (0 is the default, meaning no macro will run at power-up). A common usage for a power-on macro is to set the baud rate to a value other than the default of 115200.
- 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:** 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 | name> = 1` through 254 (or name) sets the power-on macro to the specified macro.
`<option> = 0` means display the power-on copyright, and 1 means do not display it.
- Example:** `*PONMAC 2`

PREPEND SCROLLING TEXTBOX

- 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 SCROLLING TEXTBOX

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.

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.

QUERY TEXT WINDOW

Description: Query a Textwindow for the number of lines and characters that fit in the viewport based on the assigned font, as well as the size (in lines and characters) of the associated buffer.

Command: `twq <n>`

Arguments: `<n>` - Textwindow number.

Example: `twq 0`

Example reply showing a Textwindow viewport that is 5 lines of 16 characters with a text buffer of 200 lines of 120 characters:
5 16 200 120

READ FRAME BUFFER LINE

Description: Displays 640 (VGA) or 800 (WVGA, SVGA) comma separated frame buffer hex words for a given display line. Each word is four ASCII characters representing a 16-bit RGB565 value.

Command: `*FB <line>`

Arguments: `<line>` is the display line buffer from 0 to 479 (VGA, WVGA) or 599 (SVGA).

READ FROM AUX PORT

Description: Reads serial data from the AUX port. The AUX port receive buffer is 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 ("><return>"). 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[b]`

Example: `ain`

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

READ TEMPERATURE

Description: Displays temperature measured by the sensor at location U10 (center of the board, under the processor module) in degrees Centigrade. The accuracy, and the minimum step size, is approximately 1.5 degrees C.

Command: temp

Returns: <degrees>.<tenths><return>

Where <degrees> is a three character number with leading zeros as spaces and tenths is a single numeric character. The 'C' equivalent to produce this is

```
printf("%3d.%1d",degrees, tenths);
```

REDRAW ROTATED POLYGON

Description: Redraws a rotated polygon or 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
<X Orig> <Y Orig> is X/Y location for polygon.

Example: ppgr 45 100 100

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

REDRAW TEXT WINDOW

Description: Redraw an existing Textwindow viewport at the current position. Useful in case the application wants to display an alternate screen image without needing to reload the Textwindow when returning to the current screen.

Command: twr <n>

Arguments: <n> - Textwindow number.

Example: twr 1

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 memory ([see warning](#)) and issues a software reset (see above). This restores the board to factory manufactured state with the exception that the contents of the data flash memory (bitmap and macro storage) is not affected.

Note: After issuing this command, the touch screen MUST BE recalibrated using the “tc” command.

Note: After issuing this command, the screen timing configuration may be set incorrectly for your specific LCD panel. If this occurs you will need to reload the appropriate config.ini file.

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. *NOTE: Only use this if it is to be followed by a touch calibrate command.*

Command: *RT

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

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 CONFIGURATION TO SD CARD

Description: Causes a copy of the current configuration to be saved to the SD Card. The file generated is in the format of a standard config.ini file with the name curconf.txt. Note that any existing file of the same name on the SD card will be over written.

Command: `*getConfigAsFile`

Returns: `:writing SD File curconf.txt.`

SAVE SCREEN SHOT TO SD CARD

Description: Causes an image of the entire screen (“screen shot”) to be saved to the SD Card. This image generated is saved in the form of a BMP file format. The name of the file is SLCD####.BMP, where “####” is the decimal value indicating a count of presently saved files minus one, up to 1,000 (i.e.: first file is SLCD0000.BMP).

Command: `*getScreenAsBMPFile`

Note: The write performance of SD Cards can vary greatly. Saving an entire screen shot can take as long as 40 seconds.

Returns: `:writing SD File SLCD####.BMP... (1 dot added each half-second or so to indicate progress; at completion, returns a prompt)`

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:** <numlines> is limited to the number of pixels in the axis of the scroll.
(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 two lines of text and then scrolls up the text area such that the lower line replaces the upper line.

SET AUX ESCAPE

- Description:** The control port can be selected by sending three consecutive special characters on the inactive port. For example, if COM0 is the active control port, sending three special characters on COM1 will cause COM1 to become the control port. This command sets the special character and stores the value in non-volatile memory ([see warning](#)); when given no arguments, it returns the current setting.
- Command:** `*auxEsc <hex value of ASCII character>`
- Arguments:** (none) - Display current escape character.
`0x01` though `0x7e` – set escape character.
- Example:** `*auxEsc 0x0d`
This sets the escape character to <return> which is the standard default value.

SET AUX PORT AUTO READ

Description: Reads or sets the AUX port auto-read flag and stores it in non-volatile memory ([see warning](#)). When set, ASCII characters that are received in the AUX port input buffer are automatically written to the main port.

Command: `auxIO [0|1]`

Example: `auxIO`

Returns: 0

Returns the current auxIO setting.

Example: `auxIO 1`

Enables AUX port auto read. When characters are received by the AUX port, they are automatically written to the main port with a preceding “a” and a trailing “\r”.

SET BAUD RATE

Description: Sets a new baud rate for COM0 or COM1 (baud0, baud1); if the port is not specified, the port currently in control will be set. This is temporary and the unit will revert to the default setting the next time power is cycled. Use a “power-on” macro to set the baud rate on power-up. Unless the command fails, the system prompt will be output at the new rate.

Command: `baud[0|1] <rate>`

Argument: `<rate>`: Valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: `baud0 57600`

This sets COM0 to 57600 baud and then outputs the system prompt.

Example: `baud 19200`

This sets COM port currently configured as the Control/Main port to 19200 baud and then outputs the system prompt.

SET BAUD RATE (Output Prompt First)

Description: Sets a new baud rate for COM0 or COM1 (baudp0, baudp1). If the port is not specified, the port currently in control will be set. This is temporary and the unit will revert to the default setting the next time power is cycled. Use a “power-on” macro to set the baud rate on power-up. The system prompt will be output at the old rate, before the rate is changed.

Command: baudp[0|1] <rate>

Argument: <rate>: Valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: baudp0 57600

This outputs the system prompt and then sets COM0 to 57600.

Example: baudp 19200

This outputs the system prompt and then sets the COM port currently configured as the Control/Main port to 19200 baud.

SET BAUD RATE (Sticky and Prompt First)

Description: Sets a new baud rate for COM0 or COM1 (baud0, baud1) and stores it in non-volatile memory ([see warning](#)). If the port is not specified, the port currently in control will be set. The system prompt will be output at the old rate, before the rate is changed.

Command: bauds[0|1] <rate>

Argument: <rate>: Valid values are 460800, 230400, 115200, 57600, 38400, 19200, 9600, 4800, and 1200.

Example: bauds0 57600

This outputs the system prompt, sets COM0 to 57600 baud, and then stores the setting in non-volatile memory, so it will be used again after a power cycle or reset.

Example: bauds 19200

This outputs the system prompt, sets the COM port currently configured as the Control/Main port to 19200 baud, and then stores the setting in non-volatile memory, so it will be used again after a power cycle or reset.

SET COLOR (Basic)

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

Command: `s <fore> <back>`

Arguments: `<fore>` = Foreground color value per the table below

`<back>` = Background color value per the table below

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

Example: `s 0 1`

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

Note: To reset the background after changing the color, the screen can be cleared using the command, 'z'. The screen is cleared to the background color.

SET COLOR (Detailed)

- Description:** Sets the background and foreground color for all commands using arbitrary RGB values. If given no argument, returns the current setting.
- Command:** S [<fore_detail> <back_detail>]
- Arguments:** <fore_detail> = Foreground color value in RGB format
<back_detail> = Foreground color value in RGB format
RGBformat = RGB444 or RGB888 format (see [Color Specifications](#))
- Note:** The recommended argument format is RGB888. The RGB444 argument is preserved for SLCD compatibility. Both arguments must be in the same format.
- Example:** S 00FF00 112233
Foreground = maximum green, background = 0x11 red, 0x22 green, 0x33 blue.
- Usage note:** To visibly change the screen to the background color after using this command, the screen must be cleared using the command, 'z'.
- Usage note:** The SLCD5 uses 565-color encoding – that is, 5 bits for red, 6 bits for green and 5 bits for blue. Therefore, the full 24-bit value specified will be mapped to the 565 space. See [RGB565 Encoding](#).

SET CONTROL PORT

- Description:** Used to set the port used to control the unit. This is stored in non-volatile memory ([see warning](#)) and will be used on power-up. Note: This switches between the MAIN and AUX ports of the PowerCom4 board.
- Command:** *com0main
This sets the main port to COM0. When the unit is powered up, COM0 will be sent the '>' prompt.
- Command:** *com1main
This sets the main port to COM1. When the unit is powered up, COM1 will be sent the '>' prompt.

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. The location is reset to (0,0) by the “z” command. If given no argument, this command returns the current setting.

Command: `sc [<x> <y>]`

Example:
`sc 10 20`
`t "hello"`

The above is equivalent to:

`t "hello" 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 bit-inverted RGB color. If given no argument, the command returns the current setting.

Command: `d [<n|x>]`

Arguments:
`n`: Normal drawing mode; draws with the colors from Set Color.
`x`: XOR drawing mode; inverts the existing pixel to draw lines.

Example: `d n`

This sets the drawing mode to normal.

Caution: In general, the XOR mode will not work as expected with pen width of more than 1. This is due to multiple pixel writes when rounded ends are drawn. The same issue arises with circles.

SET FONT

| | | |
|--------------|---|---|
| Description: | Sets the font to be used in subsequent TEXT DISPLAY and BUTTON DEFINE commands. The “f?” command will list available fonts (both built-in fonts and externally downloaded fonts, if any). | |
| Commands: | f
f? | |
| Arguments: | is one of the following.

Proportional fonts ; equivalent to Windows Arial at point size shown; trailing ‘s’ signifies not antialiased (s = “standard”):

8s, 8Bs, 8, 8B, 10s, 10Bs, 10, 10B, 12, 12B, 14, 14B, 16, 16B, 20, 20B, 24, 24B, 32, 32B, 48, 64

Monospace fonts ; equivalent to Windows Monospac821 BT at point size shown. Fonts m48 and m64 are NOT antialiased.

m8, m8B, m10, m10B, m12, m12B, m14, m14B, m16, m16B, m20, m20B, m24, m24B, m32, m32B, m48, m64 | |
| Example: | f 16B | Sets the current font to 16 point Arial bold. |
| Example: | f? | Displays a list of loaded fonts |
| Example: | f | Displays the current font |

SET or QUERY (LATCHING) STATE BUTTON

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

SET LED

- Description: The LED D2 on the board usually is controlled by the system and is on when the software is not idle. For debug and other purposes, the LED can be controlled by the host using this command.
- Command: `led [on|off|sys]`
- Arguments: `sys` is the default state.

SET ORIGIN

- Description: Sets the origin, relative to the upper left corner of the display, for all subsequent operations including lines, text, bitmaps, buttons and so forth (but not another ‘o’ command – origin cannot be nested). This is useful for macros that draw compound objects. If the macro draws everything relative to (0,0), by setting the origin before calling the macro, the compound object can be placed anywhere on the screen. If given no argument, returns the current setting. *Note that the SET CURSOR command location is relative to this global origin, AND the “z” command will reset the origin to the upper left corner.*
- Command: `o [<x> <y>]`
- Arguments: `<x>` X axis value between 0 and 319 for QVGA or 479 for VGA.
`<y>` Y axis value between 0 and 239 for QVGA or 639 for VGA.
- Example: `o 10 20`
`t "hello" 0 0`
- This sets the origin to x=10, y=20, and then displays the text “hello” at absolute location 10, 20.

SET PEN WIDTH

- Description: Sets the pen width for line drawing commands including line, rectangle but not circle. Default is width of 1 at power up, width of 2 after the [Touch Calibrate](#) command is executed. If given no argument, the command returns the current setting.
- Command: `p <pixels>`
- Arguments: `<pixels>` is a number from 1 to 200.
- Example: `p 1`
- This sets the pen width to 1 pixel wide.

SET 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 TEXT ALIGNMENT

Description: Sets the alignment of the text with respect to the insert point specified. Used with the text display command. If given no argument, the command returns the current setting. **NOTE: The horizontal alignment reverts to “L” (left) after a text display command is issued, but the vertical alignment remains as set.**

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

Arguments: The first letter argument is horizontal (Left, Center, Right), second is vertical (Top, Center, Bottom) alignment.

Example 1: `ta CC`
`t "hello" 100 100`

This draws the text “hello” centered horizontally and vertically on x=100, y = 100.

Example 2 (assume VGA LCD):

`ta RB`
`t "bottom right corner" 639 479`

This draws the text “hello” in the bottom right hand corner of the LCD screen.

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
- `b` Beep on notify (normal, use `-b` to disable beep)
- `t` Typematic
- `T` Typematic special (beep on first press only)
- `x` include relative x and y in notification (hotspots only)

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 and stores the value in non-volatile memory ([see warning](#)). With no arguments, the command returns the current setting. 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 and stores the values in non-volatile memory ([see warning](#)). With no arguments, the command returns the current settings. If the optional 's' is present, the command does NOT store values.

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. Both <delay> and <repeat> must be numbers between 0 and 255 (inclusive).

Example: `typematics 200 50`

This sets the delay to 2 seconds and the repeat rate at 500ms = 2 per second, but does not store the values, making the change temporary.

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. When setting EEPROM values, note the EEPROM chip has limited total write cycles ([see warning](#)).

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

Arguments: <internal variable name>

| | |
|------------------|---------------|
| Integer | - i0 thru i19 |
| EEPROM val | - e0 thru eF |
| 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. When setting EEPROM values, note the EEPROM chip has limited total write cycles ([see warning](#)).

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

Arguments: `<internal variable name>`
Integer - i0 thru i19
EEPROM val - e0 thru eF

Examples: `setx i9 A5`
Set internal integer variable i9 to HEX A5.

SLIDER DEFINE

| | | | | | | | | | | | | | | | | | | | | | |
|---------------------|--|------------------|--|-----------------|-------------------------|-------------------|--|---------------------|--|------------------|--|-------------------|---|------------------|--|-------------------|----------------------------------|-----------------|--|-----------------|--|
| Description: | Creates a slider object using background and slider control bitmaps. | | | | | | | | | | | | | | | | | | | | |
| Command: | <code>sl idx bg x y slider off ornt inv cont hi lo</code> | | | | | | | | | | | | | | | | | | | | |
| Arguments: | <table><tr><td><code>idx</code></td><td>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.</td></tr><tr><td><code>bg</code></td><td>Background bitmap index</td></tr><tr><td><code>x, y</code></td><td>Top left corner to place the background bitmap</td></tr><tr><td><code>slider</code></td><td>Slider control (e.g. knob / button) bitmap index</td></tr><tr><td><code>off</code></td><td>Slider offset from the edge of the background bitmap</td></tr><tr><td><code>ornt</code></td><td>Orientation: 0 = vertical; 1 = horizontal</td></tr><tr><td><code>inv</code></td><td>Invert: 0 = top / left is low; 1 = bottom / right is low</td></tr><tr><td><code>cont</code></td><td>Always one. Value has no impact.</td></tr><tr><td><code>hi</code></td><td>Maximum slider value, negative numbers allowed</td></tr><tr><td><code>lo</code></td><td>Minimum slider value, negative numbers allowed</td></tr></table> | <code>idx</code> | 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. | <code>bg</code> | Background bitmap index | <code>x, y</code> | Top left corner to place the background bitmap | <code>slider</code> | Slider control (e.g. knob / button) bitmap index | <code>off</code> | Slider offset from the edge of the background bitmap | <code>ornt</code> | Orientation: 0 = vertical; 1 = horizontal | <code>inv</code> | Invert: 0 = top / left is low; 1 = bottom / right is low | <code>cont</code> | Always one. Value has no impact. | <code>hi</code> | Maximum slider value, negative numbers allowed | <code>lo</code> | Minimum slider value, negative numbers allowed |
| <code>idx</code> | 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. | | | | | | | | | | | | | | | | | | | | |
| <code>bg</code> | Background bitmap index | | | | | | | | | | | | | | | | | | | | |
| <code>x, y</code> | Top left corner to place the background bitmap | | | | | | | | | | | | | | | | | | | | |
| <code>slider</code> | Slider control (e.g. knob / button) bitmap index | | | | | | | | | | | | | | | | | | | | |
| <code>off</code> | Slider offset from the edge of the background bitmap | | | | | | | | | | | | | | | | | | | | |
| <code>ornt</code> | Orientation: 0 = vertical; 1 = horizontal | | | | | | | | | | | | | | | | | | | | |
| <code>inv</code> | Invert: 0 = top / left is low; 1 = bottom / right is low | | | | | | | | | | | | | | | | | | | | |
| <code>cont</code> | Always one. Value has no impact. | | | | | | | | | | | | | | | | | | | | |
| <code>hi</code> | Maximum slider value, negative numbers allowed | | | | | | | | | | | | | | | | | | | | |
| <code>lo</code> | 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: `1128:50`

SLIDER VALUE

| | | | | | |
|------------------|---|------------------|--------------|------------------|----------------------|
| Description: | Sets the value of a previously defined slider object. | | | | |
| Command: | <code>sv idx val</code> | | | | |
| Arguments: | <table><tr><td><code>idx</code></td><td>Slider index</td></tr><tr><td><code>val</code></td><td>Value for the slider</td></tr></table> | <code>idx</code> | Slider index | <code>val</code> | Value for the slider |
| <code>idx</code> | Slider index | | | | |
| <code>val</code> | Value for the slider | | | | |
| Example: | <code>sv 128 50</code>
Sets slider index 128 to value 50. | | | | |

SPEAKER ON / OFF (Rev D and later boards only)

Description: Turns the external speaker on/off and stores the setting in non-volatile memory ([see warning](#)) to be restored on power-on. Normally, the external speaker is off and the onboard beeper is on; when the speaker is turned on, the beeper is turned off. If no setting is given, the command returns the current setting.

Command: `*spkr [on|off]`

Arguments: `[on| off]` external speaker setting.

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”, which will be shown at power-on instead of the copyright notice and firmware version text string; the setting is stored in non-volatile memory ([see warning](#)). When given no arguments, the command returns the current setting (0 is the default, meaning ‘off’).

The Windows program BMPload.exe is used to download bitmaps into the SLCD5 data flash memory. See [Appendix D](#) for details.

Command: `*SPL <number>`

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

Example `*SPL 5`

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

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 two 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
```

This displays the text:

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

with the top left corner of the 'P' at location x=10, y=0, in Normal mode.

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

Displays the text

```
©Copyright  
1999-2009
```

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

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

Displays the text (in Normal Text Mode)

```
This is in a box
```

centered in a rectangle with word wrap enabled; "This is in" is the first line; "a box" is the second line; the rectangle is at 100, 100 is 61 pixels wide and 101 pixels tall.

TEXT FLASHING DELETE

- Description: Deletes the specified text flash animation.
- Command: `tfx <index>`
- Arguments: `<index>` is the identifier for the text; accepted identifiers are 0 through 9.
- Examples: `tfx 0`
This stops and deletes from memory the specified (0) text animation.

TEXT FLASHING DISABLE

- Description: Disables a flashing text animation as specified by the index (see the [TEXT FLASHING DISPLAY](#) command). The stopping point state can be specified.
- Command: `tfd <index> <state>`
- Arguments: `<index>` is the identifier for the animation; valid range is 0-9.
`<state>` Specifies the state to stop the animation, for text flash, this is 0 or 1.
- 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.
- 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.

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 current foreground color, then erased (by drawing in the background color) at a rate specified by the parameter <t> (delay time). Each alternate view is displayed for <t> mS. The total time to cycle is 2X the selected delay time. See [TEXT DISPLAY](#) for text string escapes and other details.

Command: `tf index t "text string" x y [R|N]`
All parameters except index and "text string" are optional. If unspecified, <t> will default to 500 mS, [x y] will be the current character position, and [R | N] will be N (Normal). 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.
- R - Reverse flashing text: The string is shown in reverse text mode, and then erased.
- N - Normal flashing text: The string is shown in normal text mode, then erased. This is the default mode of operation, and need not be specified.

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

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

This puts the text `FLASHING TEXT` with the top left corner of the 'F' at location x=10, y=0 with a delay of 300 milliseconds between displayed and non-displayed text.

Note: The clear screen command 'z' clears all flashing text instances (and all other types of animations).

TEXT FLASHING ENABLE

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

Command: `tfe <index>`

Arguments: <index> is the identifier for the 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: `tf s`

Arguments: None.

Examples: `tf s`

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. **Note: pen width gets changed to 2.**

Command: `tc`

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

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

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 AND pushing the button does not cause a beep. This is useful when the macro contains an OUT command to generate arbitrary button responses.

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

Arguments: See [TOUCH MACRO ASSIGN](#).

Example: `xmq 5 2`

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

TOUCH MACRO ASSIGN WITH PARAMETERS

Description: Links a button or hotspot to a parameterized macro. When the button or hotspot is touched, the associated macro is executed with the specified arguments. If given no arguments, returns the amount of free argument storage space.

Note: The maximum number of arguments, and the maximum size of each argument, is version-dependent (see [Appendix E](#)).

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

Arguments: <t> the index of the button or hotspot.

 <action> is one of:

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

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

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

Example 1: bd 1 100 100 1 "test" 10 15
 xa 1 p 17 Check

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

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

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

```
t 0 0 Check
```

Example 2: xa 1 p 17 Check

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

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

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

```
t 0 0 Check
```

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.

TOUCH SIMULATE MOMENTARY

Description: Simulates a momentary touch (press and release) on the touch panel at the specified coordinates.

Command: `*tsmm <x> <y>`

Arguments: `<x>` X location of simulated touch.
`<y>` Y location of simulated touch.

Example: `*tsmm 100 100`

The display module will behave as if location 100/100 was touched momentarily.

TOUCH SIMULATE PRESS

Description: Simulates a press on the touch panel at the specified coordinates.

Command: `*tsmp <x> <y>`

Arguments: `<x>` X location of simulated touch press.
`<y>` Y location of simulated touch press.

Example: `*tsmp 100 100`

The display module will behave as if location 100/100 was touched and not released.

TOUCH SIMULATE RELEASE

Description: Simulates a release on the touch panel at the specified coordinates.

Command: `*tsmr <x> <y>`

Arguments: `<x>` X location of simulated touch release.
`<y>` Y location of simulated touch release.

Example: `*tsmr 100 100`

The display module will behave as if location 100/100 was released. Has no effect if the location was not already being touched (as from a `*tsmp`).

TOUCH SIMULATE TYPEMATIC

Description: Simulates a single typematic cycle on the touch panel at the specified coordinates. If the specified location was being touched (as from a `*tsmp`), and the screen item at that location was not already in typematic mode, the display module will do a single typematic cycle/action.

Command: `*tsmt <x> <y>`

Arguments: `<x>` X location of simulated touch typematic.
`<y>` Y location of simulated touch typematic.

Example: `*tsmt 100 100`

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 of the software.

Command: `vers`

Returns: `SLCD5-PLUS V<maj>.<min>.<bld> <resolution> <panel string>`

Parameters: `<resolution>` – this is the resolution standard of the display (VGA, WVGA, SVGA).
`<panel string>` – Contact REACH Technology with Panel String to determine display panel information.

Example: `SLCD5-PLUS V1.2.63 WVGA "VGG8048_PVLED"`

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 one second or in the case of a macro, delays execution by one second.

WAIT FOR REFRESH

Description: Imported from SLCD43/6/+ for compatibility. Similar to WAIT VERTICAL RETRACE, it works only in Landscape orientation. Returns when a vertical display retrace occurs. This command is used to avoid “tearing” or “flashing” in animations.

Command: `wrf <x> <y>`

Arguments: `<x>` Ignored in SLCD5; present for compatibility only.
`<y>` This is the y coordinate. This is also the number of horizontal scan lines to wait after retrace. Note that the horizontal scan lines go from the top to bottom of screen.

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

Example: `wrf 100 135`

The 100 is ignored (no portrait mode on SLCD5); same as `wvr 135`.

WAIT VERTICAL RETRACE

Description: Returns when a vertical display retrace occurs with an optional offset. Used to avoid “tearing” in animations.

Command: `wvr [<line>]`

Arguments: `<line>` Optional number of vertical lines to wait after retrace. Used to wait until the display trace has passed a specified vertical display line.

Example: `wvr 135`

This waits until vertical refresh plus 135 vertical lines. Useful to put in an animation script before displaying a bitmap at x,100 with height 35. Note that for best results, bitmaps for animations should be stored uncompressed in flash memory.

WINDOW RESTORE

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

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

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.

| One Area | Two Areas | Four Areas |
|----------------------------------|-------------------------|-------------------------|
| Index none or 0; max WxH=1048576 | Index 0; max WxH=524288 | Index 0; max WxH=262144 |
| | | Index 1; max WxH=262144 |
| | Index 2; max WxH=524288 | Index 2; max WxH=262144 |
| | | Index 3; max WxH=262144 |

Example: `ws 0 180 160 239`

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

6. Fonts

6.1 External Fonts

External fonts allow users to use fonts which are available in Microsoft 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 eight 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 which would like a .SIF file with individual characters (rather than a range of Unicode values) should submit a Microsoft 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 Character Set – ISO 8859-1

The ISO 8859-1 character set used by all fonts is as follows. Note: The ASCII character set is the same as the ISO set up to Code 127. The ISO set does not define characters 0-31, or 127-159.

The following tables can be used to determine the character code for non-ASCII characters. The standard ASCII set is 0x20 through 0x7E. The extended set is from 0xA0 through 0xFF.

For example, to display the copyright symbol, note that it is hex A9, so the command to display the character is:

t "\xa9"

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|------|---|---|---|---|----|---|---|---|---|---|---|---|---|---|---|---|
| 0020 | | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 0030 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 0040 | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 0050 | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 0060 | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 0070 | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | □ |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|------|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00A0 | | ı | ç | £ | ¤ | ¥ | ¦ | § | ¨ | © | ª | « | ¬ | - | ® | ¯ |
| 00B0 | ° | ± | ² | ³ | ´ | µ | ¶ | · | ¸ | ¹ | º | » | ¼ | ½ | ¾ | ¿ |
| 00C0 | À | Á | Â | Ã | Ä | Å | Æ | Ç | È | É | Ê | Ë | Ì | Í | Î | Ï |
| 00D0 | Ð | Ñ | Ò | Ó | Ô | Õ | Ö | × | Ø | Ù | Ú | Û | Ü | Ý | Þ | ß |
| 00E0 | à | á | â | ã | ä | å | æ | ç | è | é | ê | ë | ì | í | î | ï |
| 00F0 | ð | ñ | ò | ó | ô | õ | ö | ÷ | ø | ù | ú | û | ü | ý | þ | ÿ |

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:

1. Drop the three lowest bits of the Red and Blue values: ~~0b76543210~~.
2. Drop the two lowest bits of the Green values: ~~0b76543210~~.
3. Shift and OR the truncated R, G and B values into a 16-bit value in the form 0bRRRRRRGGGGGGBBBBB.

8. Using CRC'd Commands

8.1 Overview

The SLCD5 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 SLCD5 responds as expected. If, however, the CRC check fails, the SLCD5 ignores the command and returns an invalid CRC response ('#<return>').

8.2 Command Protocol

The format for a CRC'd command is:

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

A '~' (tilde) character at the start of the command string signals the SLCD5 that an embedded CRC (4 ASCII-Hex chars, [0-9,a-f,A-F]) will follow the '~' and then the actual SLCD5 command will begin. The CRC is calculated for the SLCD5 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 0x4050. Send:

```
~4050s 0 1<return>
```

The SLCD5 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.

8.3 Example CRC generation code

Included below is 'C' code for a program that accepts a standard SLCD5 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 SLCD5. The CRC polynomial is CRC-16-IBM.

```

//=====
// Functions for calculating CRC-16-IBM
// (Bisync, Modbus, USB, ANSI X3.28, CRC-16, or CRC-16-ANSI)
//=====

// CRC tables for the CRC16. The polynomial is x^16 + x^15 + x^2 + 1

unsigned char CRChi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
};

unsigned char CRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x4A, 0x4E, 0x8E, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
};

/*-----< CalcCRC >-----*/
unsigned short CalcCRC(unsigned char *ptr, unsigned int length)
{
    register unsigned char k, crclo, crchi;

    crclo = 0xff;
    crchi = 0xff;

    while (length--)
    {
        k = crclo ^ *ptr++;
        crclo = crchi ^ CRChi[k];
        crchi = CRCLo[k];
    }
    return (unsigned short)crclo + ((unsigned short)crchi << 8);
}
//=====

```

```

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

static char cmdStr[ 129 ];

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

    // 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 = CalcCRC( (unsigned char *)cmdStr, strlen(cmdStr) );
        // 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: [~4050s 0 1\r]

```

```

C:\CRC>cmdcrc "t \"Hello\""
Input: [t "Hello"\r]
Output: [~298Ct "Hello"\r]

```

9. Communications Watchdog Timer

9.1 Overview

As of version 1.3.0 and later, the SLCD5+N 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 events is detected:

1. Short Term loss of communication
 - No characters received for X seconds
 - Specified macro executes with label “:st_down”
2. Communication restored after Short Term loss
 - A character is received after a Short Term loss I
 - Specified macro executes with label “:st_up”
3. Long Term loss of communication
 - No characters received for Y seconds
 - Specified macro executes with label “:lt_down”
4. Communication restored after Long Term loss
 - A character is received after a Long Term loss I
 - 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 SLCD5 is reset.

9.2 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 .

9.3 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
```

10. Working with Variables

10.1 Overview

In versions prior to 1.2.0, the SLCD5 has supported a limited set of variables with primitive access capabilities. As of version 1.3.0, simple printf() like formatting is supported; this allows using variables in formatted output on the SLCD5+N display or in a line of text sent to one of the serial ports. As of version 1.4.12, the number of integer variables (i0-i9) has been increased to a total of 20 (i0-i19).

10.2 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 – eF | Numeric | 8-bit unsigned integer |
| s0 – s9 | String | Text string, max of 80 chars |
| p0 – p9 | Numeric | Pair of X and Y Coordinates (16-Bit integers) |

10.3 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 Comm port Baud Rate |

10.4 Formatting Variables

A simple printf()-like output format may be applied to a variable by inserting a format specifier between the first 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

`<max>` Maximum number of characters from string variable to output

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.

11. Using Simple Math Expressions

11.1 Overview

Since version 1.2.0, the SLCD5 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 1.3.0 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

11.2 Limitations and Requirements

- No nesting of expressions is allowed (to avoid problems with recursion).
- BMPload for the SLCD5, version 2.2.2 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.

11.3 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
```


Appendix A – LCD and Touch Panels Compatible with the SLCD5+N Controller

A.1 LCD Panels

The SLCD5+N can support a variety of LCD panels. The main issue is cabling and the backlight driver interface.

Reach manufactures custom flat flex cabling for interfacing the SLCD5+N to Hitachi and other panels that use 40 pin FFC connectors. For panels with LED backlights, the SLCD5+N has an on-board LED backlight driver. See [Appendix J](#) for schematics.

Reach also supports most panels via ERG inverters and backlight drivers; see www.ergpower.com. Please contact Reach sales for specific requirements.

For LVDS panels, the SLCD5+N controller supports panels that support 3 LVDS data streams (16-bit color) with resolutions up to SVGA (800x600).

A.2 Resistive Touch Panels (hardware version dependent)

The on-board resistive touch controller supports 4 wire resistive panels with either 1mm pigtail contact spacing (Fujitsu 4 wire standard), or 0.1” spacing.

A.3 Projected Capacitive Touch Panels (hardware version dependent)

The on-board PCAP touch controller supports the Evervision 7” panel P/N VGG804808-6UFLWL.

Appendix B – Parts and Suppliers for SLCD5+N Connections

B.1 Connectors and Cables for J6, J7, J13

The board connector is Molex type 53261-XX71 where XX is the number of pins. The mating connector is made of two parts: the receptacle housing and the crimp pins. A special tool is needed to make the crimps. Alternatively, custom cables can be purchased. See below for cable vendors.

J6 Receptacle housing Molex P/N 51021-0800

J7 Receptacle housing Molex P/N 51021-1000

J13 Receptacle housing Molex P/N 51021-0700

Crimp pins Molex 50079 or 50058

Prototype (small qty) crimp tool Molex 63811-0200

Production crimp tool Molex 63811-0000

All of the above are available from www.digikey.com

B.2 Connector for J2

J2 mating connector: DF19G-20S-1C(05)

B.3 Connectors for J11, J12

J11 mating connector: JST BHSR-02VS-1

J12 mating connector: JST BHR-02(8.0)VS-1N)

B.4 Mating Cable for J10

J10 mating cable: FFC 10-pin 0.5mm pitch, 0.3mm thick

B.5 Discrete Wire Cable Vendors

The cables needed for J6 through J13 can be specified and supplied as assembled cables by:

International Component Technology

www.intcomptech.com

Appendix C – Ordering Information

C.1 General Information

See www.reachtech.com for SLCD5+N controller and display model ordering numbers. When ordering either a board or a module, you must also specify a specific two-part revision number. The first number indicates the hardware version and accounts for options such as touch connector configuration. The second number designates the firmware that comes pre-loaded on the board. The firmware revision numbering ensures that production boards have the same firmware as was tested and approved.

C.2 Contact Reach Directly for Specific Ordering Information

Reach Technology, Inc.
4575 Cushing Parkway
Fremont, California 94538
(510) 770-1417

Appendix D – BMPload Program

D.1 Overview

The BMPload program is used to load bitmaps, optional macros, and optional fonts into the SLCD5+N controller. It creates a compressed binary file containing all these elements. This file is either loaded directly into the SLCD5+N flash via a serial port, or stored on an SD card that is inserted into the controller. There are different use cases depending on whether the controller is being used for development or as part of the end equipment production. See [System Firmware Boot Process](#) for a discussion of the exact sequence power-on loading of the binary file.

Small Image Development

If the bitmaps and other files are relatively small, BMPload can be connected to the SLCD5+N controller via a serial port, and the binary file directly loaded into the controller flash memory. This is useful for rapid testing of different images and macros.

Large Image Development

When the binary file is too large for the serial programming method (loading is too slow), BMPload can store the file on an SD card. This card is then inserted into the SLCD5+N controller, and it will be read from the card into DRAM on power-on. This loading process can delay the controller power-on but in most cases this is acceptable for development.

Alpha Test Through Production

In production, the file used in Large Image Development is placed on an SD card along with a file named “FLASH2.INI” whose presence directs the controller to store the binary file into the on-board flash memory. Once written to flash, the SD card does not need to be used, and the power-on to screen active time is short.

D.2 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.

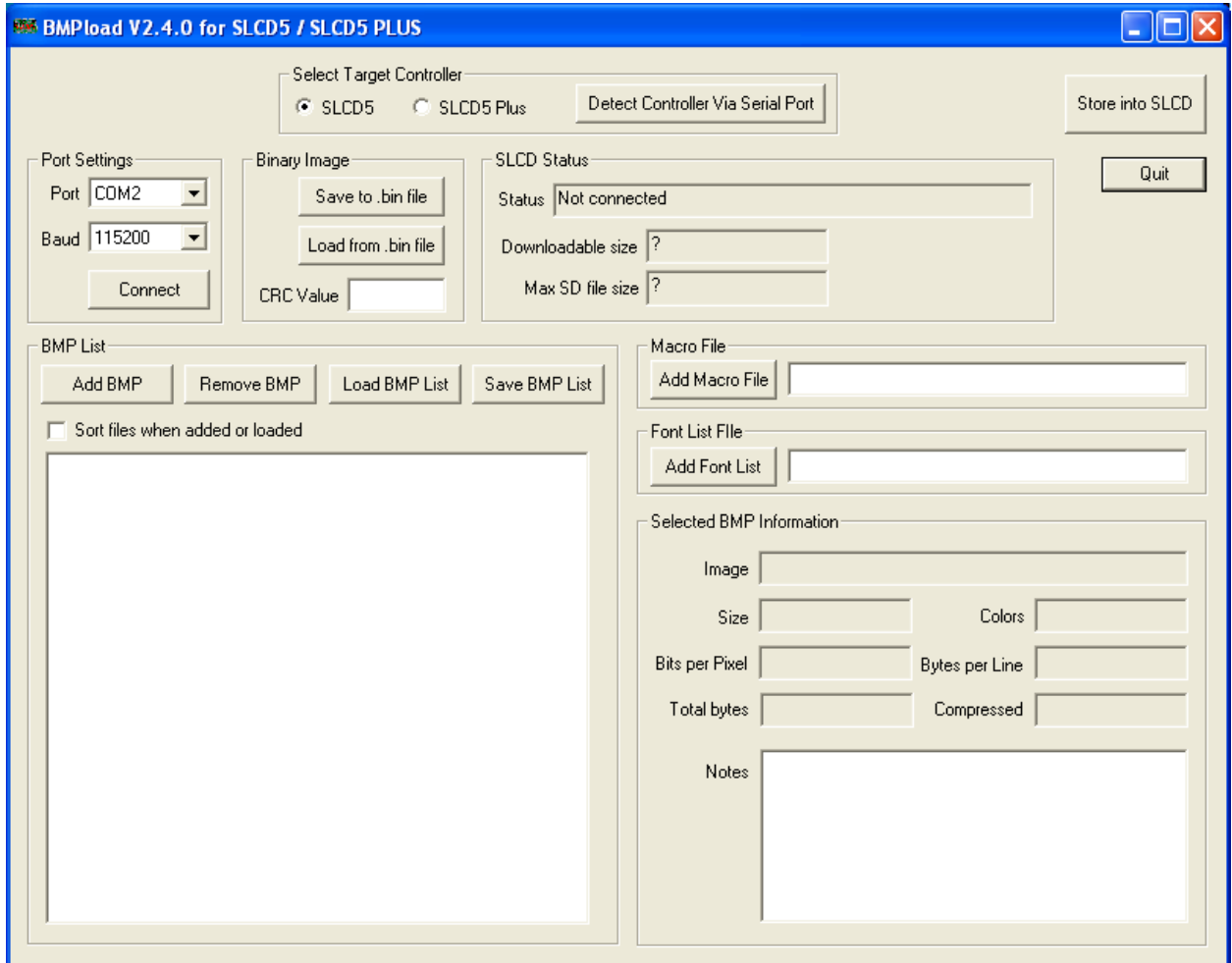
D.3 Bitmap Format

The BMPload program requires bitmaps to be in 24-bit RGB color. The SLCD5+N uses 16-bit color in RGB565 format; the BMPload program does the required translation.

BMPload also compresses images using RLE16 (run length encoding). This is very efficient for control surface images that have horizontally repeated pixels of the same color.

D.4 Program Operation

BMPload runs under Windows XP through Windows 7. The computer running BMPload must have a serial port connected to the SLCD5+N board using either the Main or Aux ports. The serial port must not be in use by another program. When it is first run, you will see the following:



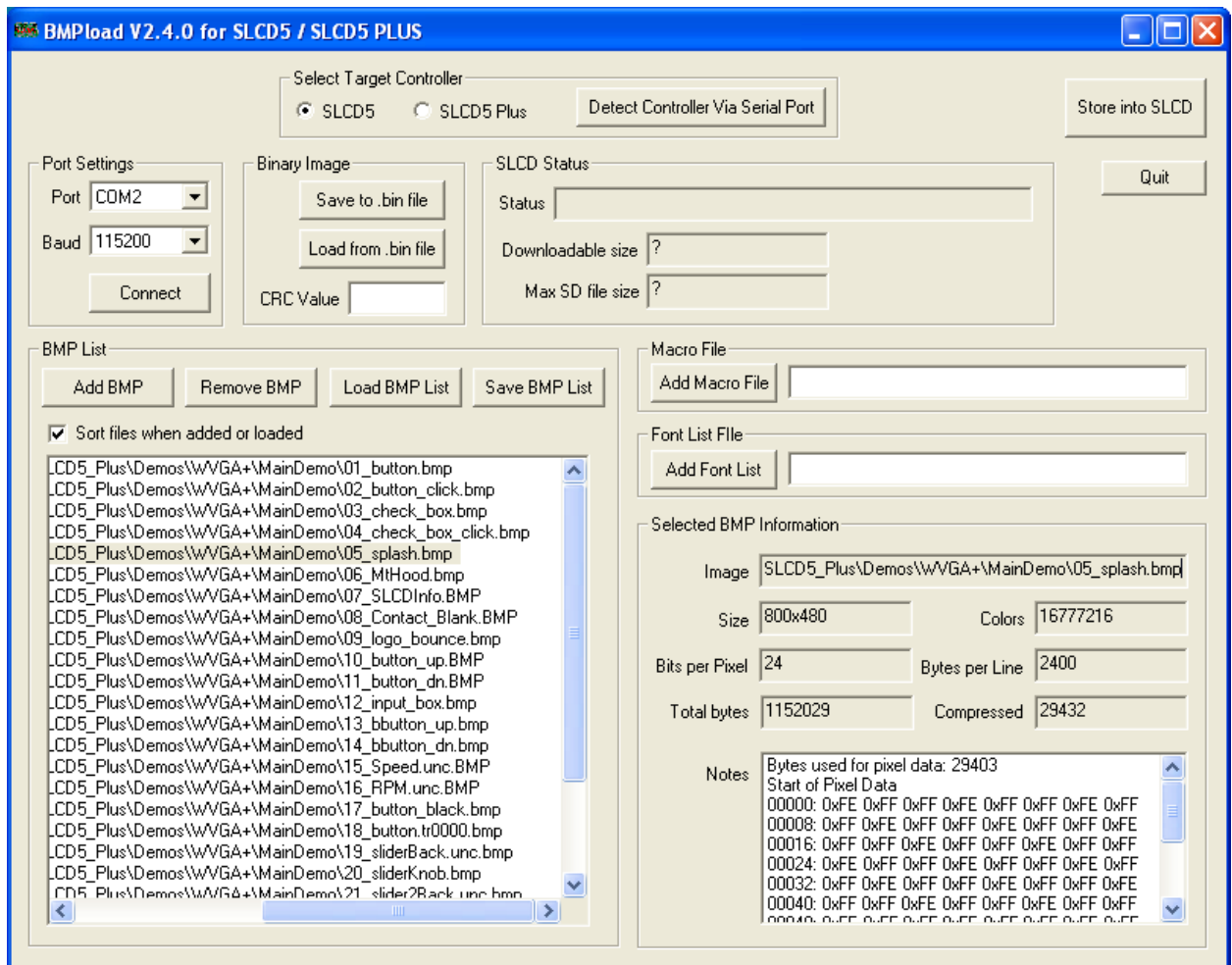
You must first set the Target Controller to the appropriate type for your SLCD5+N controller board. If you are already connected to the serial port, the type can be automatically detected and set by clicking the “Detect Controller Via Serial Port” button.

You can now use the “Add BMP” button to add BMP files to the list. Note that the order is important because you use the index number in the DISPLAY BITMAP IMAGE command. The best way to keep this clear is to start all bmp file names with their index number, for example “01_first_bitmap.bmp” and then enable the “Sort files when added or loaded” checkbox.

A list file (.lst) is a better way to load all the image files used for the interface. It also keeps the file names in the correct order. The list file is a standard text file; a list file with the following files was used for the screen shot that follows. Note that since the full path name is not specified, the files are assumed to be in the same directory as the list file.

| | |
|------------------------|---------------------------|
| 01_button.bmp | 14_bbutton_dn.bmp |
| 02_button_click.bmp | 15_Speed.unc.BMP |
| 03_check_box.bmp | 16_RPM.unc.BMP |
| 04_check_box_click.bmp | 17_button_black.bmp |
| 05_splash.bmp | 18_button.tr0000.bmp |
| 06_MtHood.bmp | 19_sliderBack.unc.bmp |
| 07_SLCDInfo.BMP | 20_sliderKnob.bmp |
| 08_Contact_Blank.BMP | 21_slider2Back.unc.bmp |
| 09_logo_bounce.bmp | 22_slider2Knob.bmp |
| 10_button_up.BMP | 23_SlidingCompass.unc.BMP |
| 11_button_dn.BMP | 24_PanelInfo.BMP |
| 12_input_box.bmp | 25_LEDOff.unc.bmp |
| 13_bbutton_up.bmp | 26_LEDON.unc.bmp |

Note that the full path name is displayed in the file list, but does not need to be in the list file. After the list has been loaded, this is what the program looks like. Note that when a bitmap is selected, the compression and size information is shown on the right.



Use the “Add Macro File” button to add a macro file to the binary file in addition to bitmaps. Once you have added the bitmaps, use the “Save to .bin File” button to save them to the SD card. The file name must be 8 characters, start with “slcd” and have the extension “.bin”.

Alternatively, you can use the “Store into SLCD” to write the binary file directly into the SLCD5+N flash memory via the serial port. This can be much slower than using the SD card.

D.5 Connecting Via Serial Port

To connect to the SLCD5+N via a serial port, select the port and / or baud rate. If the specified port is in use by another program, you will see something like this (COM1 will be replaced by the COM port you are trying to open)



This means that the program was unable to open the specified port. This is due to another program such as Tera Term being open and connected to it. Shut down or disconnect any other serial programs, and click OK.

You may see the following message instead:



This means that the serial connection between the PC and the SLCD5+N is not working. Check the cables.

D.6 BMPload Speed Issues

For fast serial loading, speeds of up to 460800 baud are supported. For reliable communication at this speed, use a USB-serial adapter with the electronics in the DB9 connector.

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

If you are using an FTDI adapter (www.ftdichip.com), you can set the latency timer to a smaller than default value to speed up transfers. To do this on an XP machine, open the Control Panel, open System, select Hardware, Device Manager, Ports, Serial port (select the USB adapter), Properties, Port Settings, Advanced, Latency Timer.

Appendix E – Macro Commands and File Format

E.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 and other graphical objects so that by pushing a button, a macro can be invoked for example to draw a new screen. This is useful to keep the overhead on the processor low and provide fast response for users.

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

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**

E.2 Macro File Format

The macro file is an ASCII text file and can be generated by Windows applications such as Notepad. There are two macro definition versions to choose from when creating 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.

The BMPload program generates a header file 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. In version 2 the name can also be used instead of the macro number when using a command that references a macro, such as the command that assigns a macro to a button.

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 text starting with the “//” string. All lines outside of a “#define...#end pair are treated as comments.

Version 2 referencing examples:

```
#define example_a
m example_b
#end
```

```
#define example_b
*PONMAC example_a
#end
```

Also with BMPload version 2.2 or higher it is acceptable to indent lines.

E.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 argExample  
t ``0`` `1` `2`  
#end
```

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

```
>m argExample Hello 10 20
```

E.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 macros that execute this way (as a result of touching a graphic object with an associated macro) will cause any currently executing macro to quit running. This also halts any macros that called the currently running macro.

E.5 Special Macro Commands, and Macro Labels

Memory commands

Memory commands were added to implement the keyboard in the demo macros that come with the SLCD5 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.

Special Arguments

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

Simple Math on Integer Arguments

``(`0`+ 1)``

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

Memory Variable

``M``

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

Integer Variable

``i0` thru `i19``

This is replaced by a 32-bit signed integer. See the [SET VARIABLE](#) command.

String Variable

``s0` thru `s9``

This is replaced by a character string with a maximum length of 80 characters. See the [SET VARIABLE](#) command.

Point Coordinate Variable

``p0` thru `p9``

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

Slider Value

``L<index>``

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

Random Number

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

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

Repeat Command

A special macro directive allows a macro to repeat execution. The directive must be at the start of a line and is:

```
:repeat
```

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

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 (the colon in column 1).

Example:

```
:beep
```

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 name/number in a macro call. For example, to invoke label “attach” in macro number 8, the command would be:

```
m 8:attach (invoking the macro by number)
```

or, if its name is “button_action”:

```
m button_action:attach (invoking the macro by name)
```

The format of a macro label invocation is: “m <macro name/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 a Label

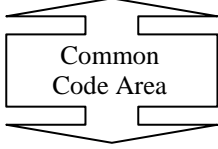
```
m 8:attach (user calls macro number 8 with label
":attach")

#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 */
```

A diagram showing a rectangular box with the text "Common Code Area" inside. The box is enclosed within a larger, double-lined rectangular frame. Four arrows point outwards from the corners of the inner box towards the corners of the outer frame, indicating the extent of the code area.

Example of Macro Call without a Label

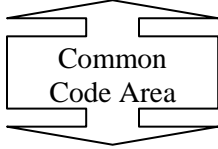
```
m 8 (user calls macro number 8 without any label)

#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.

E.6 Changing The Firmware 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 SLCD5 and run the command:

```
*PONMAC 1
```

Now cycle power to the SLCD5, and the initial baud rate will be 9600 baud.

E.7 Changing the Firmware Power-on Baud Rate

The power on macro can change the firmware baud rate, or the [SET BAUD RATE \(Sticky\)](#) command can be used to change the setting in EEPROM. Note that the bootloader will start at 115200 baud.

Appendix F – Troubleshooting

F.1 Touch Unreliable or Non-Operative

If the touch screen is unreliable or non-operative, do the following:

1. Make sure the metal shell of the display is connected to one of the SLCD5+N mounting holes. This is the same as saying that the display case should be grounded to SLCD5+N ground. Note that this only applies to displays with CCFL backlights, not to LED backlights.
2. Run the [TOUCH CALIBRATE](#) command, “tc”. This will reset the calibration values and allow you to recalibrate the touch screen.

If after doing this, the touch is still non-operative, check the touch connection into the SLCD5 board. Some touch panels use conductive ink that can be easily scraped off by too many or incorrect connector insertion cycles. If there are holes you can see through on the touch connector end where it plugs into the SLCD5+N connector this is the problem.

To determine the accuracy and sensitivity of the touch, you can use the “touch debug” command as follows:

```
*debug 1
```

This puts an “X” on the screen whenever a valid touch is recognized. To turn it off, use:

```
*debug 0
```

Appendix G – Evaluation / Demo Kit Tutorial

G.1 Self-Running Demonstration

The kit comes shipped with a demo installed. This tutorial will uninstall the demo. To run the demo afterwards, place the supplied SLCD.BIN and RUNDEMO.INI files in the root directory of an empty SD card, install the card, and reset the board or cycle power. This will run the demo. Note that it may be slow to load because it is big, but in production on the SLCD5+N, it will be loaded into flash for fast startup.

G.2 Connection and Control Via PC

This section describes how to connect to and control the SLCD5 from a PC type computer. Any other computer (Mac / Unix / Linux) can be used instead with analogous procedures.

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

The PC needs to run a serial terminal emulator. Options are Realterm or Teraterm, both are open source. Assuming Teraterm, once installed do the following:

Setup -> Serial Port -> Baud Rate 115200, Flow control Xon/Xoff

Setup -> Terminal -> Local Echo checked, Newline Receive: CR+LF

Setup -> Save Setup -> Save (replaces default setup file)

In the case of a USB-serial adapter, you may have to open the Control Panel -> System -> Hardware -> Device Manager -> Ports to determine which COM port is mapped to the USB adapter. With FTDI adapters, the device Properties -> Port Settings -> Advanced allows you to assign a specific COM port to the adapter.

Once this is done, open the terminal emulator, connect to the Main DB9 port, and hit enter. You should see a '>' prompt come back. Now, type 'z' followed by the enter key. The display should clear as a result. You should also see the 'z' letter you typed and the '>' prompt. You are successfully controlling the SLCD5 now.

G.3 Simple Commands

This section presents some simple commands that illustrate some of the SLCD5 capabilities.

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, and then paste into the terminal screen.)

Clear the screen:

```
z
```

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

```
f 24B // set font command
t "Hello World" 100 110 // draw text command
```

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

```
z
f 24B
s 16 2 // set foreground and background color
t "Hello World" 100 110
```

Restore default colors (black text, white background) and clear screen"

```
s 0 1
z
```

Create a solid pure blue rectangle with corners (140, 200) and (160, 250)

```
r 140 100 160 250 1 0000FF
```

Define momentary pushbutton #1 named "Test" at location (100,50) which is the top left corner of the button, use default (internal) bitmaps for the button, and send notification when both pressed and released:

```
z
f 16B
bdc 1 150 110 5 "Test" // button define command
//press button on screen to see notification messages)
```

Create types of hotspot

```
z
p 5 // set pixel width 5 for rectangle command
r 100 200 150 250
x 128 100 200 150 250 // standard type
//press rectangle on screen to see notification messages
r 200 250 300 300
xst 129 200 250 300 300 // invisible, typematic, only first touch beeps
```

G.4 Bitmaps

Note: If you have another serial port available, hook it up to the AUX DB9 so that bitmaps and macros can be downloaded without having to disconnect the terminal emulator. Otherwise, when using BMPload, make sure the terminal emulator is disconnected.

Start the BMPload program. Select the controller type or use the "DETECT CONTROLLER button. Click "Load BMP list", and look in the supplied CD for the Demo directory containing bitmaps (.BMP) and list files (.lst) for the screen size you are using. For the 7" screen, that is WVGA. Click on the demo.lst and a set of bitmap files will appear in the main window. Note that the files are numbered; this is useful because bitmaps are referred to by load order, so the number helps identify the bitmaps.

Clicking on any bitmap file name will display statistics for the bitmap. You may have to scroll horizontally to see the bitmap as full path names are shown. For this next tutorial section, it is assumed that image #25 is 25_LED OFF.unc.bmp and 26 is 26_LED ON.unc.bmp.

Make sure the Serial port is configured to the correct speed and port. Click on “Connect” to test the connection. If connected, click on “Store into SLCD” and it should download the bitmaps to the controller. If the download is less than 8KB/sec, and you are using a USB-serial adapter, you may have to reduce the latency; see BMPload Speed Issues in [Appendix D](#). Note that serial download is suitable for small bitmaps and for larger file collections you need to use the SD card method described in the Fonts Section below.

Once loaded, use Teraterm to connect to the controller. Do the following:

Display the bitmaps loaded in the controller:

```
ls
```

Display a bitmap by number:

```
xi 5 0 0 // display bitmap #6 with top right corner at 0,0
```

Use “LED” bitmaps for the two states of a latching button with index #2:

```
z
bdc 2 100 200 2 "" "" 25 26
// touch red dot to see touch notification with latched state
```

G.5 Macros

The easiest way to develop and investigate the uses of macros is to have a small set of bitmap images and a small macro file. See the demo macro file macros.txt for examples of macros.

G.6 Fonts

See Section 6 for a discussion of font lists and files. This tutorial will also demonstrate how to use an SD card with BMPload. For conversion between Unicode and UTF-8, see <http://www.utf8-chartable.de/>

Restart BMPload, and select “Add Font List”, and open the fonts.rfl file from the same directory as the bitmaps used above. Then insert a blank SD card into the PC and hit the Save to “.bin file” button and save the file as slcd.bin on the SD card root directory. Then exit the BMPload program and eject the SD card via the File Explorer right click menu. Install the card into the SLCD5 and reset the board. Then do the following:

List available fonts:

```
f?
// font names are listed; last one is CH48 which was just downloaded
```

Set font and display

```
z
utf8 on // enable UTF-8 to address more than 256 characters
f 16B
o 100 50 // use an offset for the text
t "UNICODE character 4E1C in 48 point is:\n"
f CH48 // select the downloaded font (alias specified in fonts.rfl)
t "\xe4\xb8\x9c" // use escapes to send hex in ASCII
(the above can be sent with the UTF-8 sent as 8 bit bytes instead of escape codes)
```

G.7 Attaching to the Embedded Controller

The PC connection method described above is useful for trying out the development kit. After that, the kit should be attached to the OEM's embedded microcontroller for actual GUI development and testing. Attaching one port (MAIN) to the microcontroller, and the other (AUX) to the micro allows either the terminal emulator or the BMPload program to share control with the micro. This is useful when a series of commands is to be tested and it is faster to test using a terminal than writing and compiling microcontroller code. To switch between the micro being the active host and the PC being the active host, either one sends three <return> characters in a row. BMPload does this and then switches the port back so the microcontroller does not have to reestablish its port as the control port. It can also be done at the terminal via the *prevCons command.

This tutorial assumes the host microcontroller has an RS232 port. This is often the case with microcontroller development kits.

Attach the micro's RS232 port to the Main SLCD5+N port; typically via a serial crossover cable. To verify, look at the voltages on pins 2 and 3 of each DB9; connect one side's pin X at a negative voltage to the other side's pin Y of no voltage and vice versa.

Power-cycle the SLCD5+N. Write code on the micro to poll the serial port waiting for the '>'<return> prompt. When it gets this, have the micro send commands such as described above and verify that they work as expected.

Appendix H – Working with Bitmaps

H.1 Creating Bitmaps

Bitmaps are used to create the visual elements of the user interface. These include buttons, tabbed folders, and data entry and display areas. Most interface styles implemented on Microsoft Windows can be duplicated on the SLCD5+N. 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: Bitmaps must be saved in 24-bit RGB color mode.

H.2 High Color

The SLCD5+N supports high color depth (RGB565) as standard. 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.

H.3 Gradients

Many modern-looking graphic designs use 24-bit color gradients. These can end up with a “banded” look when converted to 16-bit color. To eliminate this effect, use a spatial diffusion converter (dither) to limit the number of colors to 16-bit while diffusing the step changes spatially. This should be done after the image is rendered for use.

A Photoshop plugin called DepthDither can be used to perform this function. Newer versions of PhotoShop include a ‘dither’ function. A web search for “dither an image in photoshop” will return many articles.

For GIMP users, there is free GIMP plugin called “16-bit (was ARGB4444) Dither Script” that will do this for you. See <http://registry.gimp.org/node/25275>.

H.4 Transparency

High Color depth 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 “.unc” indicates that the bitmap is uncompressed, which is required for transparency. When this bitmap is displayed, the transparent color will not be displayed.

Appendix I – SD Card Support Features

I.1 Overview

The SLCD5+N board contains an SD card slot. The SD card can be used to do the following:

- Initialize non-volatile (EEPROM) system variables via config.ini file (see [System Configuration](#)).
- Load new firmware into the SLCD5 controller.
- Hold a BMPload binary file (bitmaps, macros, fonts) to be used as the working set of images, macros, and fonts
- Update the on-board flash with the above binary file
- Save screen shots.
- Run demo macro on startup.
- Hold multiple BMPload binary files (working sets) to be loaded as needed
- Hold individual image files of type .jpg, .gif, and .bmp

Note: The SD card must be 2GB or less in size and formatted as FAT (also known as FAT16) to work properly with the SLCD5. **Do not use FAT32.** Some SD cards must be reformatted before use. Reach recommends the SanDisk brand of SD card.

I.2 Firmware Upgrade

On power-on, the SLCD5+N boot loader program checks if an SD card is present, and if so, looks in the root directory for a file with the name:

SLCD????.ELF

where ‘?’ is any character. If it finds this file, it compares the file name with the name of the last loaded firmware stored in non-volatile memory (EEPROM). If they are different, it initiates a firmware update. This process may take a minute or two, during which time the LED D2 flashes, and progress messages appear on COM0 (115200 baud). Once the firmware has been upgraded it starts running.

To force an update (ignore the stored name), place a non-null text file called “LOADFW.INI” in the root directory.

I.3 Holding Binary Working Set File

When the SLCD5+N controller powers on, it looks to load a binary file (working set of images, macros, and fonts). If an SD card is present, and has a file named SLCD*.BIN, this file is used. It is loaded into DRAM because loading images on the fly from SD is too slow. Loading into DRAM provides the fastest display time for images, but this loading can delay the power-on of the display. This can be eliminated by writing the binary file to on-board flash.

I.4 Update Binary File in Flash

In order to speed up the power-on display time, the binary file can be loaded into on-board flash. To do this, include a non-null text file called “FLASH2.INI” in the SD card root directory. When the SLCD5+N controller powers on, if an SD card is present, and has a file named SLCD*.BIN,

and the appropriate FLASH2.INI is present, the binary file is directly loaded into flash memory and used from there. This may take some time; the main serial port will display progress dots, the LCD screen will display status messages and dots, and the on-board LED will flash.

1.5 Screen Shot Saving

Users can save the current contents of the entire screen on a file on an inserted SD Card. Refer to the [SAVE CONFIGURATION TO SD CARD](#) command for details.

1.6 Run Demo Macro

A special file, rundemo.ini can be used to launch a macro on power-on. If an SD card contains both a bitmap/macro binary file (e.g. SLCD.BIN) and this special file, a demo can be run without disturbing the binary file stored in flash.

The rundemo.ini file only has two options: verbose mode and the number of the macro to run. Verbose mode is the same as for the config.ini file (See [System Configuration](#)). To specify the macro to run, use the syntax below.

```
Demomac = <macro number>
```

Example:

```
demomac = 1
```

1.7 Multiple Binary Files

In some cases it is useful to be able to switch the “working set” of images, macros, and fonts. This could be done to support different GUI styles (skins) or languages. This can be done by storing the binary files in different root directories on the SD card. The command “*sdload <directory name>” can be used to load a different binary file. See [LOAD FILE FROM SD CARD \(SPECIFIED DIRECTORY\)](#).

1.8 Separate Image Files

The SD card can also be used to hold individual images as .bmp, .jpg, or .gif formatted files. This may be useful if, for instance, a user’s manual was to be stored and displayed page by page. The “xif” command is used to display these files. The “*cd” command changes the current directory on the SD card so that different groups of images can be used. See [CHANGE SD CARD DIRECTORY](#).

Appendix J – LED Backlight Driver Schematics

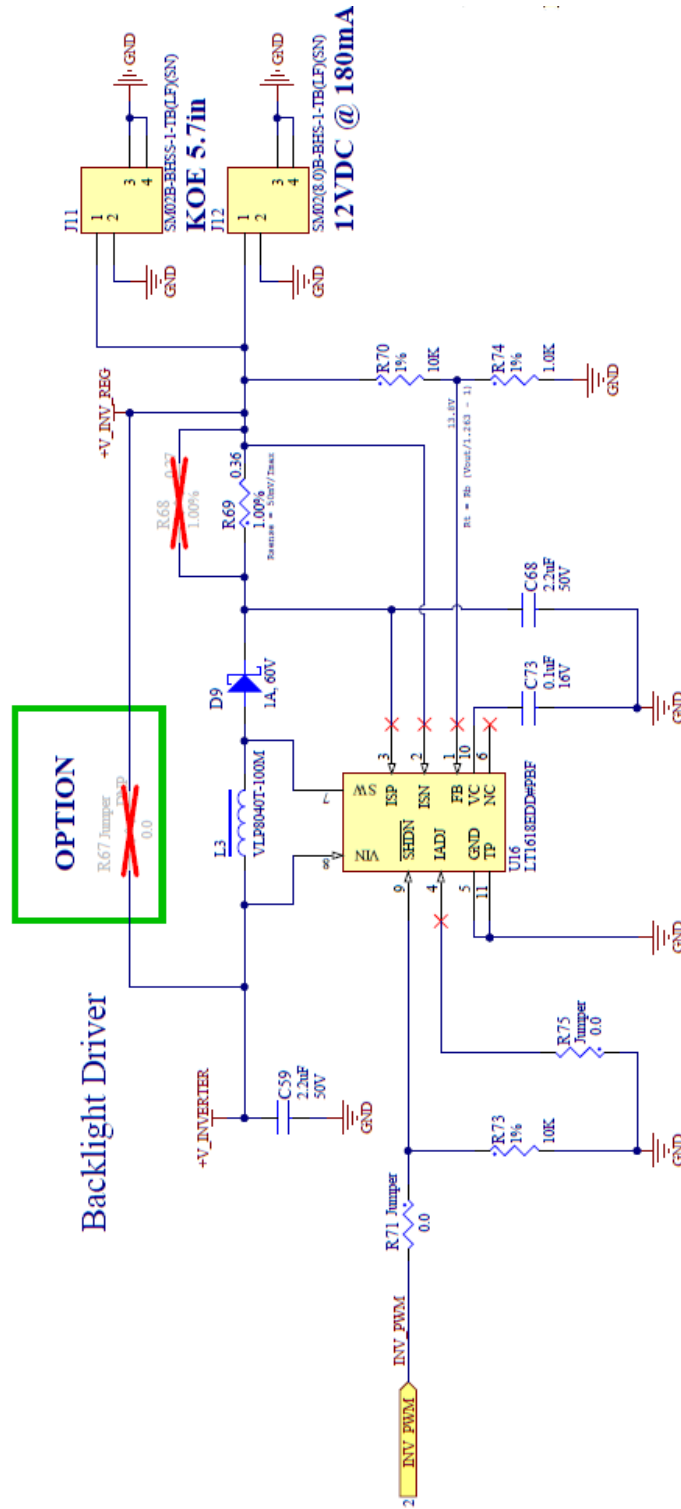


Figure 11: LED Backlight Driver Schematics

Appendix K – Board Version Differences

Differences from SLCD5/SLCD5+ Products

This section details the major changes made from the SLCD5/SLCD5+ products.

- SLCD5+ microprocessor daughter-board integrated into SLCD5 base board.
- Connector J8 removed (old VGA TFT panel connector).
- Connector J10 removed (external backlight driver).
- Connectors J11, J12, and J12A removed (resistive touch sensor connectors).
- Connector J14 removed (external SD card slot connector).
- Header JP4 removed ('inverter default off' jumper).
- Header JP5 removed (unused feature).
- LCD buffers removed to reduce noise emissions.
- LED D1 changed name to D3, function remains the same (VCC OK).
- LED D1 on daughter board is now D1 on the integrated board, function remains same (internal use, 'executing code').
- COM0 on J6 now automatically switches between 3.3V CMOS mode or RS232 mode, based on harness wiring. Stuff option to force mode removed.
- COM1 on J7 now automatically switches between 3.3V CMOS mode or RS232 mode, based on harness wiring. No longer need to use an external wire from pin 4 to pin 9.
- Connector J1 added (resistive touch sensor).
- Connector J10, J17 added (I2C PCAP sensor).
- Connectors J11, J12 added (backlight power)
- Connector J2 added (LVDS).
- Connector J5, pin 3 is now connected to ground.

42-0323-001

This version is configured with a parallel LCD interface (33-pin FFC connector), with the backlight current set to 140ma for a 5.7" LCD panel (R68 not installed).

42-0323-101

This version is configured with a parallel LCD interface (33-pin FFC connector), with the backlight current set to 200ma for a 7" LCD panel (R68 installed).

42-0323-301

This version is configured with an LVDS LCD interface, and a resistive touch panel.

Appendix L – Manual Change History

| <i>Date</i> | <i>Changes</i> |
|-------------|---|
| 12/01/2015 | Adapted from SLCD5/SLCD5+ Manual. |
| 12/03/2015 | First release. |
| 12/08/2015 | Fixed typo in Features list, removed J10 pinout. |
| 02/18/2016 | Added model/board version information, added board pictures. Clarified connector information. |
| 02/29/2016 | Corrected GET/SET command descriptions to reflect integer variable range up to i19. |