Next Generation Intelligent LCD Panels

Command Documentation
for DPC10xx / DPC20xx / DPC3020 iLCD Controller

Version 3.41
Document Date: December 2, 2009

Copyright © by demmel products 2004 - 2009

Unless otherwise noted, all materials contained in this document are copyrighted by demmel products and may not be used except as provided in these terms and conditions or in the copyright notice (documents and software) or other proprietary notice provided with the relevant materials.

demmel products
hard & software

demmel products ■ Radnitzkygasse 43 ■ A-1100 Vienna/Austria ■ Phone +43-1-6894700-0 ■ Fax+43-1-6894700-40 ■ Email: office@demmel.com ■ Web: www.demmel.com

iLCD Command Documentation

Index

Important Chapters to Read

We fully understand that you probably won't want to read the whole of this document before starting to use your iLCD. Nevertheless, we strongly recommend that you read the chapters listed below as these will give you a good grounding in how to communicate with an iLCD controller. This will help you later on when you come to construct your own command strings and macros.

"Command Structure" on page 1
"Syntax Used in Setup Program" on page 2
"Command Overview" on page 3
"The Concept of iLCD's Touch Fields" on page 42 when you want to use touch fields

The iLCD setup software mentioned in the chapter "Syntax Used in Setup Program" below contains some examples with comments (use the "Load… button on the screen "iLCD Terminal") allowing you to carry out your first steps. Besides that you can load the miscellaneous demo files via the "Load…" button on the "iLCD Setup" screen and have a look at the well commented macros by pressing "Edit → Macros…" to learn more about how to communicate with iLCD.

Scope of Document

The iLCD product family consists of three generations of controller basically sharing the same set of commands. To find out which controller your iLCD panel contains, use the iLCD setup software and click on the "Response Test…" button on the "Preferences" section after having connected your iLCD panel with the PC.

A few commands are specific to certain iLCD controllers or some of the common commands may have a different format depending on the controller. In this manual the differences between controller types are depicted using the symbols shown below. Commands not marked with these symbols can be assumed to be common to all controller types.

1 Available for DPC10xx iLCD controller only
2 Available for DPC20xx iLCD controller only
3 Available for DPC3020 iLCD color controller only

The DPC3080 color iLCD controller uses a further advanced command set, which is a superset of the commands for the DPC3020. When using this controller, please refer to the documenetation found on http://www.demmel.com/download/ilcd/ilcd-color-commands.pdf

Command Structure

All commands must start with a special command introducer (written as **<CID>**), which has the value Hex AA, and the command character followed by the data block. If the data block contains a **<CID>** character, it must be quoted by a preceding **<CID>** (Hex AA) character. This allows the controller to resynchronize in case of data errors or hang-ups of the controlling application.

All commands respond with an <ACK> (Hex 06) character after they have been processed. If the command is unknown or contains an invalid value, a <NACK> (Hex 15) is returned as soon as the invalid value is recognized and the controller returns to "sync mode", which means it waits for a new **<CID>** character. In this case, the source of the problem is saved as an error code and can be accessed via the command "Get Last Error Code" (page 18). A complete list can be found in the Error Codes table on page 66. If an unexpected **<CID>** character is found, processing of the current command is cancelled, and the next character is tried to be interpreted as a command. Any characters after a successful interpreted command, which are not a **<CID>** character, are simply ignored.

The iLCD controller has connections to the serial port's RX, TX, CTS, RTS and GND pin only; the RTS pin is ignored. Please note that the controlling application has to monitor the CTS pin to avoid overflowing the controller's buffer, which has a size of 128 bytes. Normally, all commands can be sent, even without monitoring the CTS pin when the application waits for the responding <ACK> character, but we recommend using the hardware handshake if possible. All commands sent to the iLCD controller fit into this 128-byte buffer. Special exceptions are the "Write Text" (page 30) and the "Get Text Extent" (page 21) commands, which may contain more than the 128 bytes of command text. If no CTS monitoring is used, the user has to take care to break the text to be outputted into smaller chunks of data and to wait for the <ACK> character after each block of data to avoid overflowing the input buffer.

When using RS-422 or RS485 no monitoring of CTS is available therefore the user must take care not to allow the input buffer to be overrun.

If a buffer overflow occurs, the controller finishes any currently running output sequence and then sends an OVR flag (Hex 19). This is not done when the communication runs via I^2C; an extra special overrun bit is set in this case.

The serial port of the controlling application has to be set to the baud rate selected via the setup program (initially 115200 baud); 8-bit, no parity and 1, 1.5 or 2 stop bits. When using the second serial port with the RS-422/RS-485 mode, 2 stop bits must be selected to allow the controller to switch the transmitter off in time.

Syntax Used in Setup Program

The iLCD setup program contains a terminal emulation allowing you to enter all possible commands used for iLCD controllers in an easy-to-use way. Some simple terminal emulation programs such as Hyperterminal, supplied with most versions of Windows, do not allow you to enter non-ANSI characters like the<CID> character (Hex AA as mentioned above) and so trying to use such programs is not advisable.

The iLCD terminal emulation uses several sequences allowing you to enter any Hex bytes or words very easily. The most important of these, perhaps, is \i representing Hex AA which precedes all command strings. If you need to remember any of the command sequences at any time, simply press the "CmdChars…" button on the screen "iLCD Terminal" to display this list. Pressing the "Commands…" button on the same screen will give you similar information on all of the iLCD commands.

Seq	Name	Hex-Value	Description
\b	<BS>	08	Back-Space Character
\e	<ESC>	1B	Escape Character
\d...	-	00 … FF	Output Decimal Byte (e.g. \d123 or \d-123)
\D...	-	0000 … FFFF	Output Decimal Word (e.g. \D12345 or \D-1)
\i	<CID>	AA	Command Introducer Hex AA
\n	<LF>	0A	Line Feed
\r	<CR>	0D	Carriage Return
\s	-	20	Space Character
\t	<TAB>	09	Tabulator Character
\x...	-	00 … FF	Output Hex Byte (e.g. \x7B)
\X...	-	0000 … FFFF	Output Hex Word (e.g. \XAB12)
\ddd	-	00 … FF	Output Decimal Byte ddd (eg. \123 or \-123)
\0	-	00	Output 0-Character
{...}	-	-	Comment, any characters between '{' and '}' are not sent

Example:

When entering the following text in the iLCD terminal emulation

```
\i!                 { Reset All}
\iCK\D100\D50       { Set Pixel Coordinate to decimal x=100, y=50 }
\iDTHello World!\0  { Write Text "Hello World!" at x=100, y=50 }
```

and pressing the "Send" button, the following Hex characters are sent:

Hex Bytes	Representation
AA 21	.!
AA 43 4B 00 64 00 32	.CK.d.2
AA 44 54 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 00	.DTHello World!.

The Hex output to be sent actually can be seen in the terminal emulation by checking the "Show Hex Command" checkbox in a similar way as shown above.

! Please note that, if you enter the decimal word value `\D170` as a parameter in a command this will cause the actual command sequence to be terminated. This is because \D170 represents Hex AA and the controller will interpret it as such. The controller will then interpret the following bytes as the beginning of a new command with unexpected results. To get around this you must 'quote' the value by entering it twice.

So, for example, if you want to set the screen coordinates to x=170, y=50, you should instead enter

```
\iCK\0\170\170\D50
```

The Hex AA character is quoted in this way to tell the iLCD controller not to start with a new command sequence but to use the AA character as a parameter instead according to the chapter "Command Structure" on page 1 .

Terminal Mode

The terminal mode allows the controlling application to send data to the iLCD module as if it were a standard ANSI controlled terminal. All keystrokes scanned by the iLCD module are reported as simple ASCII characters, so using the terminal mode enables the user of, for example, a Linux system to redirect the standard console to the iLCD module.

The iLCD modules can be switched into a terminal mode via a command sequence (see "Go Terminal Mode" on page 23). They can even start in terminal mode when set to this mode using the setup program. When in terminal mode, no commands can be sent to the iLCD until terminal mode is exited via the "End terminal mode" escape sequence (see "Private ANSI extensions on page 16). In terminal mode, keystrokes are not reported via the make and break key, anymore; only the key code is reported. See chapter "The Concept of iLCD's Touch Fields" on page 42 and "Enable Keyboard Report" on page 50 for detailed information.

If the controlling application cannot monitor the CTS pin (see above) the iLCD module can instead work with XON/XOFF control characters (see "Set XON/XOFF For Terminal Mode" on page 24), the XON/XOFF mode can be turned on at startup when set via the setup program. Please note that XON/XOFF mode is only active when running in terminal mode.

Command Overview

16-Bit Values

Some commands (mainly those commands dealing with coordinates) require a 16-bit word describing the parameter. All these 16-bit values must be sent in the form of high byte first and low byte next. That means that to send a value such as decimal 345 (= Hex 159) you must send Hex 01 followed by Hex 59.

In the following description of the commands, the parameters of a 16-bit value are always written as `xxx_hb` and `xxx_lb` where `xxx` describes the 16-bit parameter.

Example:

```
<CID> C C address_hb address_lb
```

The describing text refers to the 16-bit value `address` (meaning `address_hb` * 256 + `address_lb`).

Copyright © by demmel products, Document Date: December 2, 2009

32-Bit Values

A very few commands require a 32-bit value describing the parameter. All these 32-bit values must be sent in the form of four bytes - high byte first and low byte last. That means that to send a value such as decimal 115200 (= Hex 0001C200) you must send Hex 00, 01, C2 and 00.

In the following description of the commands, the parameters of a 32-bit value are always written as `xxx_b3 xxx_b2 xxx_b1 xxx_b0` where `xxx` describes the 32-bit parameter.

Example:

```
<CID> I B port_no value_b3 value_b2 value_b1 value_b0
```

The describing text refers to the 32-bit value (meaning `value_b3` * 16,777,216 + `value_b2` * 65,536 + `value_b1` * 256 + `value_b0`).

16-Bit Color Values

The DPC3020 iLCD controller supports commands for setting colors such as background color or foreground color. Although the DPC3020 iLCD controller internally works with 16-bit color values, all commands except the read and write scan line commands use 24-bit color values (see at "24-Bit Color Values" on page 5) to allow future expansions. The read and write scan line commands use the 16-bit color values only to minimize the amount of data to be read and written, as one 320x240 color pixel screen needs 320*240*2 = 153,600 bytes to be read/written even in 16-bit color format.

The bit assignment of the 16-bit color values is R5G6B5, which make up one 16-bit word as follows:

`RRRRRGGGGGGBBBBB` (most significant bit is leftmost)

The formula (C-notation) to get a 16-bit color value from the red/green/blue parts is as follows:

```
color_16_bit = (red << 11) + (green << 5) + blue
```

where `red` and `blue` has a range of 0…31 and `green` has a range of 0…63, the maximum value for `red/green/blue` refers to 100% color intensity. Shifting `red` left by 11, shifting `green` left by 5 and adding the shifted red and green value and blue together delivers the 16-bit color value.

Some color values shown in binary and hex representation explain the color values further:

Color	Red (Dec. / %)	Green Dec. / %)	Blue (Dec. / %)	Binary Value	Hex Value
Pure red	31 / 100%	0 / 0%	0 / 0%	1111100000000000	F800
Pure green	0 / 0%	63 / 100%	0 / 0%	0000011111100000	07E0
Pure blue	0 / 0%	0 / 0%	31 / 100%	0000000000011111	001F
Black	0 / 0%	0 / 0%	0 / 0%	0000000000000000	0000
White	31 / 100%	63 / 100%	31 / 100%	1111111111111111	FFFF
Yellow	31 / 100%	63 / 100%	0 / 0%	1111111111100000	FFE0
Magenta	31 / 100%	0 / 0%	31 / 100%	1111100000011111	F81F
Cyan	0 / 0%	63 / 100%	31 / 100%	0000011111111111	07FF

In the following description of these commands, the parameters of a 16-bit color value are always written as `xxx_hb` and `xxx_lb` where `xxx` describes the 16-bit color parameter.

Copyright © by demmel products, Document Date: December 2, 2009

Example:

<CID> D N W no_of_pixels_hb no_of_pixels_lb p0_hb p0_lb p1_hb p1_lb ...

This is the command for writing a (partial) scan line with no_of_pixels length made up of pixels p1, p2, etc. An example showing the hex parameters for setting a red, a green and a green pixel starting from the current cursor position is as follows:

<CID> D N W 00_H 03_H $F8_H$ 00_H 07_H $E0_H$ 00_H $1F_H$

24-Bit Color Values

The DPC3020 iLCD controller supports commands for setting colors such as background color or foreground color. Although the DPC3020 iLCD controller internally works with 16-bit color values, all commands except the read and write scan line commands use 24-bit color values to allow future expansions.

The 24-bit color value is converted to a 16-bit color value by the iLCD controller internally by using the following formula:

```
color_16_bit = ((red >> 3) << 11) + ((green >> 2) << 5) + (blue >> 3)
```

where red, green and blue are 8-bit values with a range between 0 and 255.

The color values must be set in a 24-bit format where color_r describes the 8-bit red part, color_g the green and color_b describes the green and blue part of the 24-bit color value.

Example setting the background color:

<CID> A C B color_r color_g color_b

The describing text refers to the 24-bit color value color.

Signed Values

Some commands take a signed value as an argument. To calculate the value to be sent, do as follows:

Signed Bytes

Signed bytes have a value range from decimal –128 … +127. If the value is positive, the hex value is the simple hex representation of the decimal value; if it is negative the calculation can be done as follows:

Hex value = hex(256 – abs(value))

This gives a hex value of 80H for decimal –128 and a hex value of FFH for decimal –1. The decimal value -100 gives a hex value of 9CH.

Signed Words

Signed words have a value range from decimal –32768 … + 32767. If the value is positive, the hex value is the simple hex representation of the decimal value; if it is negative the calculation can be done as follows:

Hex value = hex(65536 – abs(value))

This gives a hex value of 8000H for decimal –32768 and a hex value of FFFFH for decimal –1. The decimal value -10000 gives a hex value of D8F0H.

Copyright © by demmel products, Document Date: December 2, 2009

Please note that signed words have to be sent in the order: high byte first, low byte next (same as for standard 16-bit values).

Commands sorted by functionality

The following commands are documented in the sections below:

Command Section	Command Description	Page	Command
General	No Operation	16	<CID> '
General	Reset All	16	<CID> !
General	Reset All and Show Startup Graphic	17	<CID> $
General	Reboot Panel Controller	17	<CID> #
General	Get Last Error Code	18	<CID> ? E
General	Get Firmware Info	18	<CID> ? I
General	Get Identification Info	18	<CID> ? M
General	Get Firmware Version	18	<CID> ? V
General	Get Serial Number	18	<CID> ? S
General	Get iLCD Controller Name	19	<CID> ? C
General	Get Hardware Revision	19	<CID> ? H
LCD-Control	Set Screen Orientation	19	<CID> C O orientation
LCD-Control	En/disable ANSI	19	<CID> C A on_off
LCD-Control	Set Page Address	20	<CID> C P address
LCD-Control	Set Column Address	20	<CID> C C address_hb address_lb
LCD-Control	Increment/Decrement Column Address	20	<CID> C c addr_inc_hb addr_inc_lb
LCD-Control	Set Row Address	20	<CID> C R address_hb address_lb
LCD-Control	Increment/Decrement Row Address	20	<CID> C r addr_inc_hb addr_inc_lb
LCD-Control	Set Pixel Coordinate	20	<CID> C K x_hb x_lb y_hb y_lb
LCD-Control	Get Pixel Coordinate	21	<CID> C ? K
LCD-Control	Get Text Extent	21	<CID> C ? T char1 char2 ... null
LCD-Control	Get Text Message Extent	21	<CID> C ? t index_hb index_lb
LCD-Control	Set Text Alignment	21	<CID> C T mode widht_hb width_lb height_hb height_lb
LCD-Control	Set Line Style	22	<CID> C L style
LCD-Control	Get Display Size	23	<CID> C ? D
LCD-Control	Set TAB Spacing	23	<CID> C S tab_spacing
LCD-Control	Set Auto-Linefeed	23	<CID> C F on_off
LCD-Control	Set Wrap Mode	23	<CID> C W horz_wrap vert_wrap
LCD-Control	Go Terminal Mode	23	<CID> C G =
LCD-Control	Set XON/XOFF For Terminal Mode	24	<CID> C X on_off
LCD-Control	Set Backlight Mode	24	<CID> C B mode
LCD-Control	Get Backlight Mode	24	<CID> C ? B
LCD-Control	Set Backlight Blink Frequency	24	<CID> C b frequency
LCD-Control	Set Backlight Intensity	25	<CID> C I intensity
LCD-Control	Get Backlight Intensity	25	<CID> C ? I
LCD-Control	Get Fixed LCD Contrast/Gamma	25	<CID> C ? X
LCD-Control	Set LCD Contrast	25	<CID> C N value

Copyright © by demmel products, Document Date: December 2, 2009

LCD-Control	Get LCD Contrast	26	<CID> C ? N
LCD-Control	Set LCD Gamma Value 3	26	<CID> C M value
LCD-Control	Get LCD Gamma Value 3	26	<CID> C ? M
LCD Attributes	Set Font	26	<CID> A F number_hb number_lb
LCD Attributes	Set Font Spacing	26	<CID> A S x_spacing y_spacing
LCD Attributes	Set Symbol Font	27	<CID> A Y on_off
LCD Attributes	Set Bold Mode	27	<CID> A B on_off
LCD Attributes	Set Underline Mode	27	<CID> A U on_off
LCD Attributes	Set Underline Position	27	<CID> A u position
LCD Attributes	Set Inverse Mode	27	<CID> A I on_off
LCD Attributes	Set Transparent Mode On/Off	28	<CID> A T on_off
LCD Attributes	Set Foreground Color 3	28	<CID> A C F red green blue
LCD Attributes	Set Background Color 3	28	<CID> A C B red green blue
LCD Attributes	Set Border Color 3	28	<CID> A C R red green blue
LCD Attributes	Set Border Shadow Color 3	29	<CID> A C S red green blue
LCD-Draw	Erase Display	29	<CID> D E
LCD-Draw	Erase Display Area	29	<CID> D e width_hb width_lb height_hb height_lb
LCD-Draw	Invert Screen	29	<CID> D I
LCD-Draw	Invert Screen Area	29	<CID> D i width_hb width_lb height_hb height_lb
LCD-Draw	Write Text	30	<CID> D T char1 char2 ... null
LCD-Draw	Write Text Message	30	<CID> D t index_hb index_lb
LCD-Draw	Scroll Up Screen	30	<CID> D S U scroll_y_hb scroll_y_lb
LCD-Draw	Scroll Down Screen	30	<CID> D S D scroll_y_hb scroll_y_lb
LCD-Draw	Scroll Left Screen	30	<CID> D S L scroll_x_hb scroll_x_lb
LCD-Draw	Scroll Right Screen	31	<CID> D S R scroll_x_hb scroll_x_lb
LCD-Draw	Read Graphics Byte 1 2	31	<CID> D ? R
LCD-Draw	Read Multiple Graphics Byte 1 2	31	<CID> D ? r count_hb count_lb
LCD-Draw	Write Graphics Byte 1 2	31	<CID> D W N byte
LCD-Draw	Write Multiple Graphics Bytes 1 2	31	<CID> D w N count_hb count_lb byte1 byte2 ...
LCD-Draw	Binary OR Graphics Byte 1 2	31	<CID> D W O byte
LCD-Draw	Binary OR Multiple Graphics Bytes 1 2	32	<CID> D w O count_hb count_lb byte1 byte2 ...
LCD-Draw	Binary AND Graphics Byte 1 2	32	<CID> D W A byte
LCD-Draw	Binary AND Multiple Graphics Bytes 1 2	32	<CID> D w A count_hb count_lb byte1 byte2 ...
LCD-Draw	Binary XOR Graphics Byte 1 2	32	<CID> D W X byte
LCD-Draw	Binary XOR Multiple Graphics Bytes 1 2	32	<CID> D w X count_hb count_lb byte1 byte2 ...
LCD-Draw	Write Scan Line 3	33	<CID> D N W no_of_pixels_hb no_of_pixels_lb p0_hb p0_lb p1_hb p1_lb ...
LCD-Draw	Read Scan Line 3	33	<CID> D N R no_of_pixels_hb no_of_pixels_lb
LCD-Draw	Set/Clear Pixel	33	<CID> D P on_off

Copyright © by demmel products, Document Date: December 2, 2009

LCD-Draw	Set/Clear Pixel At X/Y	33	`<CID> D p x_pos_hb x_pos_lb y_pos_hb y_pos_lb on_off`
LCD-Draw	Draw Line	34	`<CID> D L end_x_hb end_x_lb end_y_hb end_y_lb`
LCD-Draw	Draw Rectangle	34	`<CID> D R mode width_hb width_lb height_hb height_lb`
LCD-Draw	Draw Circle	34	`<CID> D C radius_hb radius_lb`
LCD-Graphics	Display Local Graphic	34	`<CID> G graph_idx_hb graph_idx_lb`
LCD-Graphics	Load Animated Graphics	35	`<CID> g L anim_loc index_hb index_lb`
LCD-Graphics	Set Coordinates To X, Y	35	`<CID> g K anim_loc pos_x_hb pos_x_lb pos_y_hb pos_y_lb`
LCD-Graphics	Set Coordinates To Current Screen Coordinates	35	`<CID> g k anim_loc`
LCD-Graphics	Start Or Restart Animation	36	`<CID> g S anim_loc`
LCD-Graphics	Stop And Set Frame Number	36	`<CID> g F anim_loc frame_hb frame_lb`
LCD-Graphics	Stop (Break) Animation	36	`<CID> g B anim_loc`
LCD-Graphics	Stop (Break) All Animations	36	`<CID> g B A`
LCD-Graphics	Set Repetitions	36	`<CID> g R anim_loc repeat_hb repeat_lb`
LCD-Graphics	Erase Image Area	36	`<CID> g E anim_loc`
LCD-Graphics	Erase Frame Area	37	`<CID> g e anim_loc`
LCD-Graphics	Suspend Animation Engine	37	`<CID> g s`
LCD-Graphics	Resume Animation Engine	37	`<CID> g r`
LCD-Screen Memory	Get # Of Screen Memory Positions	37	`<CID> M S ?`
LCD-Screen Memory	Save Screen To Memory	38	`<CID> M S S index`
LCD-Screen Memory	Recall Screen From Memory	38	`<CID> M S C index`
LCD-Screen Memory	Paint Stored Screen	38	`<CID> M S P index`
LCD-Screen Memory	Invert Stored Screen	38	`<CID> M S I index`
LCD-Screen Memory	Scroll Up Stored Screen	38	`<CID> M S U index scroll_y_hb scroll_y_lb`
LCD-Screen Memory	Scroll Down Stored Screen	39	`<CID> M S D index scroll_y_hb scroll_y_lb`
LCD-Screen Memory	Scroll Left Stored Screen	39	`<CID> M S L index scroll_x_hb scroll_x_lb`
LCD-Screen Memory	Scroll Right Stored Screen	39	`<CID> M S R index scroll_x_hb scroll_x_lb`
LCD-Screen Memory	Set Height Of Stored Screen	39	`<CID> M S H index height_hb height_lb`
LCD-Screen Memory	Set Width Of Stored Screen	39	`<CID> M S W index width_hb width_lb`
LCD-Cursor Memory	Save Cursor & Attributes To Memory	40	`<CID> M C S index`
LCD-Cursor Memory	Restore Cursor & Attributes from Memory	40	`<CID> M C C index`
Macros	Execute Macro	41	`<CID> O E index_hb index_lb`
Macros	Jump to Macro	41	`<CID> O J index_hb index_lb`
Macros	Delay Macro Execution	41	`<CID> O D delay_hb delay_lb`
Macros	Set Macro Execution Speed	41	`<CID> O S speed_hb speed_hb`
Macros	Allow Keyboard/Touch Macros To Start	41	`<CID> O K`
Macros	Set Macro Timer	42	`<CID> O T time`
Touch Screen	Calibrate Touch Screen	43	`<CID> T C`

Touch Screen	Calibrate Touch Screen and Report	43	<CID> T c
Touch Screen	Verify Touch Screen Calibration	43	<CID> T V
Touch Screen	Set Touch Field Width	44	<CID> T W width_hb width_lb
Touch Screen	Set Touch Field Height	44	<CID> T H height_hb height_lb
Touch Screen	Set Touch Field Make Macro	44	<CID> T M macro_idx_hb macro_idx_lb
Touch Screen	Set Touch Field Break Macro	44	<CID> T B macro_idx_hb macro_idx_lb
Touch Screen	Set Touch Field Text Template Index	44	<CID> T T text_template_idx_hb text_template_idx_lb
Touch Screen	Create/Define Touch Field	45	<CID> T A field_idx key
Touch Screen	Remove Touch Field	45	<CID> T R field_idx
Touch Screen	Global En/Disable Touch Fields	45	<CID> T G on_off
Touch Screen	Set Touch Field Index	45	<CID> T I field_idx
Touch Screen	Execute Touch Make Macro	46	<CID> T E M field_idx
Touch Screen	Execute Touch Break Macro	46	<CID> T E B field_idx
Touch Screen	Draw Touch Field Text	46	<CID> T D field_idx
Touch Screen	Enable/Disable Reporting Touch-Coordinates 2 3	47	<CID> T K onoff
Touch Screen	Enable/Disable Reporting Movements 2 3	47	<CID> T O onoff
Touch Screen	Retrieve Last Touch Screen Event 2 3	47	<CID> T ?
Input/Output	Set LED	48	<CID> I L S led_no mode
Input/Output	Set Multiple LEDs	48	1 <CID> I L s led_mask blink_mask 2 3 <CID> I L s led_mask_hb led_mask_lb blink_mask_hb blink_mask_lb
Input/Output	Set LED Blink Frequency	49	<CID> I L F frequeny
Input/Output	Set Relays On/Off/PWM	49	<CID> I R relay_no mode
Input/Output	Relays One Shot	49	<CID> I r relay_no mode time_hb time_lb
Input/Output	Enable Keyboard	49	<CID> I K E on_off
Input/Output	Enable Keyboard Report	50	<CID> I K R on_off
Input/Output	Get Keyboard State	50	<CID> I K ?
Input/Output	Set Baud Rate	50	1 <CID> I B divisor_hb divisor_lb 2 3 <CID> I B port_no baud_b3 baud_b2 baud_b1 baud_b0
Input/Output	Get Current Communication-Port 2 3	51	<CID> I ? C
Input/Output	Get Inputs State	51	<CID> I ? I
Input/Output	Get ADC Value	52	<CID> I ? A port
Time/Date	Set Time 2 3	52	<CID> I T hour minute second
Time/Date	Get Time 2 3	53	<CID> I ? T
Time/Date	Set Date 2 3	53	<CID> I D year month day weekday
Time/Date	Get Date 2 3	53	<CID> I ? D

PWM	Set PWM #0	54	[1] <CID> I P 00_H prescaler_hb prescaler_lb pulse_width period polarity [2] [3] <CID> I P 00_H freq_b3 freq_b2 freq_b1 freq_b0 duty_cycle_hb duty_cycle_lb
PWM	Set PWM #1	55	[1] <CID> I P 01_H duty_cycle [2] [3] <CID> I P 01_H duty_cycle_hb duty_cycle_lb
EEPROM	Get EEPROM Size [2] [3]	56	<CID> E ?
EEPROM	Erase EEPROM	56	<CID> E E =
EEPROM	Read EEPROM	56	<CID> E R index_hb index_lb
EEPROM	Write EEPROM	56	<CID> E W index_hb index_lb value
Power/Watch Dog	Set Watchdog Interval	57	<CID> P W interval_hb interval_lb
Power/Watch Dog	Trigger Watchdog	57	<CID> P w
Power/Watch Dog	Shutdown (Power Off)	58	<CID> P U =
Power/Watch Dog	Hard Shutdown (Long Power Off)	58	<CID> P u =
Power/Watch Dog	Cancel Shutdown	58	<CID> P U C
Power/Watch Dog	Get Power State	59	<CID> P ?
Power/Watch Dog	Reset Motherboard	59	<CID> P ! =
Power/Watch Dog	Set Smart Power-Off Mode	59	<CID> P S on_off
Power/Watch Dog	Set Power-Off Notification On/Off	60	<CID> P N on_off

Commands sorted by command sequence

Command Section	Command Description	Page	Command
General	No Operation	16	<CID> '
General	Reset All	16	<CID> !
General	Reboot Panel Controller	17	<CID> #
General	Reset All and Show Startup Graphic	17	<CID> $
General	Get iLCD Controller Name	19	<CID> ? C
General	Get Last Error Code	18	<CID> ? E
General	Get Hardware Revision	19	<CID> ? H
General	Get Firmware Info	18	<CID> ? I
General	Get Identification Info	18	<CID> ? M
General	Get Serial Number	18	<CID> ? S
General	Get Firmware Version	18	<CID> ? V
LCD Attributes	Set Bold Mode	27	<CID> A B on_off
LCD Attributes	Set Background Color [3]	28	<CID> A C B red green blue
LCD Attributes	Set Foreground Color [3]	28	<CID> A C F red green blue
LCD Attributes	Set Border Color [3]	28	<CID> A C R red green blue
LCD Attributes	Set Border Shadow Color [3]	29	<CID> A C S red green blue
LCD Attributes	Set Font	26	<CID> A F number_hb number_lb
LCD Attributes	Set Inverse Mode	27	<CID> A I on_off
LCD Attributes	Set Font Spacing	26	<CID> A S x_spacing y_spacing
LCD Attributes	Set Transparent Mode On/Off	28	<CID> A T on_off
LCD Attributes	Set Underline Mode	27	<CID> A U on_off
LCD Attributes	Set Underline Position	27	<CID> A u position
LCD Attributes	Set Symbol Font	27	<CID> A Y on_off
LCD-Control	Get Backlight Mode	24	<CID> C ? B

LCD-Control	Get Display Size	23	`<CID> C ? D`
LCD-Control	Set Backlight Intensity	25	`<CID> C ? I`
LCD-Control	Get Pixel Coordinate	21	`<CID> C ? K`
LCD-Control	Get LCD Gamma Value [3]	26	`<CID> C ? M`
LCD-Control	Get LCD Contrast	26	`<CID> C ? N`
LCD-Control	Get Text Extent	21	`<CID> C ? T char1 char2 ... null`
LCD-Control	Get Text Message Extent	21	`<CID> C ? t index_hb index_lb`
LCD-Control	Get Fixed LCD Contrast/Gamma [3]	25	`<CID> C ? X`
LCD-Control	En/disable ANSI	19	`<CID> C A on_off`
LCD-Control	Set Backlight Blink Frequency	24	`<CID> C b frequeny`
LCD-Control	Set Backlight Mode	24	`<CID> C B mode`
LCD-Control	Increment/Decrement Column Address	20	`<CID> C c addr_inc_hb addr_inc_lb`
LCD-Control	Set Column Address	20	`<CID> C C address_hb address_lb`
LCD-Control	Set Auto-Linefeed	23	`<CID> C F on_off`
LCD-Control	Go Terminal Mode	23	`<CID> C G =`
LCD-Control	Set Backlight Intensity	25	`<CID> C I intensity`
LCD-Control	Set Pixel Coordinate	20	`<CID> C K x_hb x_lb y_hb y_lb`
LCD-Control	Set Line Style	22	`<CID> C L style`
LCD-Control	Set LCD Gamma Value [3]	26	`<CID> C M value`
LCD-Control	Set LCD Contrast	25	`<CID> C N value`
LCD-Control	Set Screen Orientation [3]	19	`<CID> C O orientation`
LCD-Control	Set Page Address	20	`<CID> C P address`
LCD-Control	Increment/Decrement Row Address	20	`<CID> C r addr_inc_hb addr_inc_lb`
LCD-Control	Set Row Address	20	`<CID> C R address_hb address_lb`
LCD-Control	Set TAB Spacing	23	`<CID> C S tab_spacing`
LCD-Control	Set Text Alignment	21	`<CID> C T mode widht_hb width_lb height_hb height_lb`
LCD-Control	Set Wrap Mode	23	`<CID> C W horz_wrap vert_wrap`
LCD-Control	Set XON/XOFF For Terminal Mode	24	`<CID> C X on_off`
LCD-Draw	Read Graphics Byte [1] [2]	31	`<CID> D ? R`
LCD-Draw	Read Multiple Graphics Byte [1] [2]	31	`<CID> D ? r count_hb count_lb`
LCD-Draw	Draw Circle	34	`<CID> D C radius_hb radius_lb`
LCD-Draw	Erase Display	29	`<CID> D E`
LCD-Draw	Erase Display Area	29	`<CID> D e width_hb width_lb height_hb height_lb`
LCD-Draw	Invert Screen	29	`<CID> D I`
LCD-Draw	Invert Screen Area	29	`<CID> D i width_hb width_lb height_hb height_lb`
LCD-Draw	Draw Line	34	`<CID> D L end_x_hb end_x_lb end_y_hb end_y_lb`
LCD-Draw	Read Scan Line [3]	33	`<CID> D N R no_of_pixels_hb no_of_pixels_lb`
LCD-Draw	Write Scan Line [3]	33	`<CID> D N W no_of_pixels_hb no_of_pixels_lb p0_hb p0_lb p1_hb p1_lb ...`

Copyright © by demmel products, Document Date: December 2, 2009

LCD-Draw	Set/Clear Pixel	33	<CID> D P on_off
LCD-Draw	Set/Clear Pixel At X/Y	33	<CID> D p x_pos_hb x_pos_lb y_pos_hb y_pos_lb on_off
LCD-Draw	Draw Rectangle	34	<CID> D R mode width_hb width_lb height_hb height_lb
LCD-Draw	Scroll Down Screen	30	<CID> D S D scroll_y_hb scroll_y_lb
LCD-Draw	Scroll Left Screen	30	<CID> D S L scroll_x_hb scroll_x_lb
LCD-Draw	Scroll Right Screen	31	<CID> D S R scroll_x_hb scroll_x_lb
LCD-Draw	Scroll Up Screen	30	<CID> D S U scroll_y_hb scroll_y_lb
LCD-Draw	Write Text	30	<CID> D T char1 char2 ... null
LCD-Draw	Write Text Message	30	<CID> D t index_hb index_lb
LCD-Draw	Binary AND Graphics Byte 1 2	32	<CID> D W A byte
LCD-Draw	Binary AND Multiple Graphics Bytes 1 2	32	<CID> D w A count_hb count_lb byte1 byte2 ...
LCD-Draw	Write Graphics Byte 1 2	31	<CID> D W N byte
LCD-Draw	Write Multiple Graphics Bytes 1 2	31	<CID> D w N count_hb count_lb byte1 byte2 ...
LCD-Draw	Binary OR Graphics Byte 1 2	31	<CID> D W O byte
LCD-Draw	Binary OR Multiple Graphics Bytes 1 2	32	<CID> D w O count_hb count_lb byte1 byte2 ...
LCD-Draw	Binary XOR Graphics Byte 1 2	32	<CID> D W X byte
LCD-Draw	Binary XOR Multiple Graphics Bytes 1 2	32	<CID> D w X count_hb count_lb byte1 byte2 ...
EEPROM	Get EEPROM Size 2 3	56	<CID> E ?
EEPROM	Erase EEPROM	56	<CID> E E =
EEPROM	Read EEPROM	56	<CID> E R index_hb index_lb
EEPROM	Write EEPROM	56	<CID> E W index_hb index_lb value
LCD-Graphics	Stop (Break) All Animations	36	<CID> g B A
LCD-Graphics	Stop (Break) Animation	36	<CID> g B anim_loc
LCD-Graphics	Erase Image Area	36	<CID> g E anim_loc
LCD-Graphics	Erase Frame Area	37	<CID> g e anim_loc
LCD-Graphics	Stop And Set Frame Number	36	<CID> g F anim_loc frame_hb frame_lb
LCD-Graphics	Display Local Graphic	34	<CID> G graph_idx_hb graph_idx_lb
LCD-Graphics	Set Coordinates To Current Screen Coordinates	35	<CID> g k anim_loc
LCD-Graphics	Set Coordinates To X, Y	35	<CID> g K anim_loc pos_x_hb pos_x_lb pos_y_hb pos_y_lb
LCD-Graphics	Load Animated Graphics	35	<CID> g L anim_loc index_hb index_lb
LCD-Graphics	Resume Animation Engine	37	<CID> g r
LCD-Graphics	Set Repetitions	36	<CID> g R anim_loc repeat_hb repeat_lb
LCD-Graphics	Suspend Animation Engine	37	<CID> g s
LCD-Graphics	Start Or Restart Animation	36	<CID> g S anim_loc
Input/Output	Get ADC Value	52	<CID> I ? A port
Input/Output	Get Current Communication-Port 2 3	51	<CID> I ? C
Time/Date	Get Date 2 3	53	<CID> I ? D

Input/Output	Get Inputs State	51	<CID> I ? I
Time/Date	Get Time 2 3	53	<CID> I ? T
Input/Output	Set Baud Rate	50	1 <CID> I B divisor_hb divisor_lb 2 3 <CID> I B port_no baud_b3 baud_b2 baud_b1 baud_b0
Time/Date	Set Date 2 3	53	<CID> I D year month day weekday
Input/Output	Get Keyboard State	50	<CID> I K ?
Input/Output	Enable Keyboard	49	<CID> I K E on_off
Input/Output	Enable Keyboard Report	50	<CID> I K R on_off
Input/Output	Set LED Blink Frequency	49	<CID> I L F frequeny
Input/Output	Set Multiple LEDs	48	1 <CID> I L s led_mask blink_mask 2 3 <CID> I L s led_mask_hb led_mask_lb blink_mask_hb blink_mask_lb
Input/Output	Set LED	48	<CID> I L S led_no mode
PWM	Set PWM #0	54	1 <CID> I P 00_H prescaler_hb prescaler_lb pulse_width period polarity 2 3 <CID> I P 00_H freq_b3 freq_b2 freq_b1 freq_b0 duty_cycle_hb duty_cycle_lb
PWM	Set PWM #1	55	1 <CID> I P 01_H duty_cycle 2 3 <CID> I P 01_H duty_cycle_hb duty_cycle_lb
Input/Output	Set Relays On/Off/PWM	49	<CID> I R relay_no mode
Input/Output	Relays One Shot	49	<CID> I r relay_no mode time_hb time_lb
Time/Date	Set Time 2 3	52	<CID> I T hour minute second
LCD-Cursor Memory	Restore Cursor & Attributes from Memory	40	<CID> M C C index
LCD-Cursor Memory	Save Cursor & Attributes To Memory	40	<CID> M C S index
LCD-Screen Memory	Get # Of Screen Memory Positions	37	<CID> M S ?
LCD-Screen Memory	Recall Screen From Memory	38	<CID> M S C index
LCD-Screen Memory	Scroll Down Stored Screen	39	<CID> M S D index scroll_y_hb scroll_y_lb
LCD-Screen Memory	Set Height Of Stored Screen	39	<CID> M S H index height_hb height_lb
LCD-Screen Memory	Invert Stored Screen	38	<CID> M S I index
LCD-Screen Memory	Scroll Left Stored Screen	39	<CID> M S L index scroll_x_hb scroll_x_lb
LCD-Screen Memory	Paint Stored Screen	38	<CID> M S P index
LCD-Screen Memory	Scroll Right Stored Screen	39	<CID> M S R index scroll_x_hb scroll_x_lb
LCD-Screen Memory	Save Screen To Memory	38	<CID> M S S index
LCD-Screen Memory	Scroll Up Stored Screen	38	<CID> M S U index scroll_y_hb scroll_y_lb
LCD-Screen Memory	Set Width Of Stored Screen	39	<CID> M S W index width_hb width_lb
Macros	Delay Macro Execution	41	<CID> O D delay_hb delay_lb
Macros	Execute Macro	41	<CID> O E index_hb index_lb
Macros	Jump to Macro	41	<CID> O J index_hb index_lb

Copyright © by demmel products, Document Date: December 2, 2009

Macros	Allow Keyboard/Touch Macros To Start	41	`<CID> O K`
Macros	Set Macro Execution Speed	41	`<CID> O S speed_hb speed_hb`
Macros	Set Macro Timer	42	`<CID> O T time`
Power/Watch Dog	Reset Motherboard	59	`<CID> P ! =`
Power/Watch Dog	Get Power State	59	`<CID> P ?`
Power/Watch Dog	Set Power-Off Notification On/Off	60	`<CID> P N on_off`
Power/Watch Dog	Set Smart Power-Off Mode	59	`<CID> P S on_off`
Power/Watch Dog	Shutdown (Power Off)	58	`<CID> P U =`
Power/Watch Dog	Hard Shutdown (Long Power Off)	58	`<CID> P u =`
Power/Watch Dog	Cancel Shutdown	58	`<CID> P U C`
Power/Watch Dog	Trigger Watchdog	57	`<CID> P w`
Power/Watch Dog	Set Watchdog Interval	57	`<CID> P W interval_hb interval_lb`
Touch Screen	Retrieve Last Touch Screen Event 2 3	47	`<CID> T ?`
Touch Screen	Create/Define Touch Field	45	`<CID> T A field_idx key`
Touch Screen	Set Touch Field Break Macro	44	`<CID> T B macro_idx_hb macro_idx_lb`
Touch Screen	Calibrate Touch Screen	43	`<CID> T C`
Touch Screen	Calibrate Touch Screen and Report	43	`<CID> T c`
Touch Screen	Draw Touch Field Text	46	`<CID> T D field_idx`
Touch Screen	Execute Touch Break Macro	46	`<CID> T E B field_idx`
Touch Screen	Execute Touch Make Macro	46	`<CID> T E M field_idx`
Touch Screen	Global En/Disable Touch Fields	45	`<CID> T G on_off`
Touch Screen	Set Touch Field Height	44	`<CID> T H height_hb height_lb`
Touch Screen	Set Touch Field Index	45	`<CID> T I field_idx`
Touch Screen	Enable/Disable Reporting Touch-Coordinates 2 3	47	`<CID> T K onoff`
Touch Screen	Set Touch Field Make Macro	44	`<CID> T M macro_idx_hb macro_idx_lb`
Touch Screen	Enable/Disable Reporting Movements 2 3	47	`<CID> T O onoff`
Touch Screen	Remove Touch Field	45	`<CID> T R field_idx`
Touch Screen	Set Touch Field Text Template Index	44	`<CID> T T text_template_idx_hb text_template_idx_lb`
Touch Screen	Verify Touch Screen Calibration	43	`<CID> T V`
Touch Screen	Set Touch Field Width	44	`<CID> T W width_hb width_lb`

ANSI Support

All text messages sent to the iLCD can contain controlling ANSI sequences, for cursor control, attribute settings and so on, and are interpreted according to the list below. Please note that setting ANSI mode to off causes the ANSI sequences to be interpreted as normal characters.

Using cursor control sequences from the standard ANSI sequences makes sense only if you have chosen a fixed pitch font as the iLCD controller always uses the maximum character width of the selected font (which is equal to the character width of any character on a fixed size font).

If you use a sequence such as `<Esc> [ 4 D` (go left 4 characters), and you have printed the text "Hill" via a proportional font, the resulting cursor position would be too far left as the 'i' and 'l' characters take much less space than the maximum character width used.

Using ANSI cursor control sequences will mainly be used on simple character output applications, which may use the iLCD's terminal mode. A typical application is showing the console output of a Linux system. In such cases there is no problem in using a fixed pitch font. More sophisticated applications can use the private ANSI extensions or run the command mode, which support many more possibilities.

Standard ANSI Sequences

Control Characters

Character	Hex-Code	Meaning
CR	0D	Carriage Return - Return to beginning of line
LF	0A	Line Feed - Go to next line
FF	0C	Form Feed - Same as LF
VT	0B	Vertical Tab - Same as LF
BS	08	Backspace - Go back one character
TAB	09	Tabulator - Go to next tabulator position
Esc	1B	Escape - Start of ANSI sequence

Attribute Sequences

Sequence	Meaning
`<Esc> [ m`	Clear bold, underline, reverse
`<Esc> [ 0 m`	Clear bold, underline, reverse
`<Esc> [ 1 m`	Set bold attribute
`<Esc> [ 4 m`	Set underline attribute
`<Esc> [ 7 m`	Set reverse attribute

Display Control Sequences

Sequence	Meaning
<Esc> [2 J	Clear display
<Esc> [K	Erase to end of line
<Esc> [0 K	Erase to end of line

Cursor Control Sequences

Sequence	Meaning
<Esc> [<n> A	Cursor up <n> rows
<Esc> [<n> B	Cursor down <n> rows
<Esc> [<n> C	Cursor right <n> columns
<Esc> [<n> D	Cursor left <n> columns
<Esc> [<x> ; <y> H	Go to <x> <y> position
<Esc> [<x> ; <y> f	Go to <x> <y> position
<Esc> [H	Go to home
<Esc> [f	Go to home
<ESC> [6 n	Get cursor pos as <ESC> [<x> ; <y> R

Please note, that <x> <y> is used in the opposite order to that in standard ANSI sequences due to a mistake when designing the initial command set of the iLCD controller. To stay compatible with earlier versions of the firmware, no change has been made to this reversed order even though it is now acknowledged.

Save Restore Sequences

Sequence	Meaning
`<ESC> [ <n> s`	Save cursor & attributes to memory position <n> (0..7)
`<ESC> [ s`	Save cursor & attributes to memory position 0
`<ESC> [ <n> u`	Restore cursor & attributes from memory position <n> (0..7)
`<ESC> [ u`	Restore cursor & attributes from memory position 0

Private ANSI extensions

The iLCD's private ANSI extensions are not covered by an independent specification, so they will work on iLCD displays only.

All sequences start with the "<Esc> {", some commands are identical to the standard ANSI sequences.

Sequence	Meaning
<ESC> { ! e N d	End terminal mode
<ESC> { ! M	Get Model Info as <ACK> model_name <ACK>
<ESC> { ! a	Get Acknowledge as <ACK>
<ESC> { <x> ; <y> H	Go to <x> <y> pixel position (1-based)
<ESC> { <x> ; <y> f	Go to <x> <y> pixel position (1-based)
<Esc> { H	Go to home
<Esc> { f	Go to home
<ESC> { 6 n	Get cursor pixel pos as <ESC> { <x> ; <y> R (1-based)
<ESC> { <n> F	Select font number <n> (1-based, 0 = startup font)
<ESC> { <n> G	Paint graphic number <n> (1-based, 0 = startup graphics)
<ESC> { <n> s	Save cursor & attributes to memory position <n> (0..7)
<ESC> { s	Save cursor & attributes to memory position 0
<ESC> { <n> u	Restore cursor & attributes from memory position <n> (0..7)
<ESC> { u	Restore cursor & attributes from memory position 0

Command Description

This section describes all commands in detail, which can be sent to the iLCD controller.

General Commands

No Operation

```
<CID> '
```

Sending this sequence does not cause any action to be done by the panel controller, but it terminates any incomplete command (as any other command does as the <CID> character is used to synchronize the state machine again) and sends an <ACK> character. This command may also be used to check the existence of the LCD panel on the selected serial port.

Reset All

```
<CID> !
```

Resets the iLCD module. The following things are done:

Power related times are set to default
Watchdog is disabled
Smart power off

Copyright © by demmel products, Document Date: December 2, 2009

Power notification is switched on
ATX Reset and power pins are released
Shutdown is cancelled

The screen orientation is set to the default value set by the setup software.
Cursor is set to 0, 0
Text alignment is reset
Bold, inverse and underline attributes are set to off
Color values are set to default
Underline position is set to 0
All animations are stopped
Animation engine is started
ANSI mode is set to startup value
Horizontal and vertical wrapping mode is set to startup value
Auto-Linefeed is set to startup value
TAB spacing is set to startup value
Backlight blink frequency is set to startup value
XON/XOFF mode is set to startup value
Font type is reset to startup font
LCD contrast is set to the corresponding EEPROM value [1])
LCD Gamma value is set to the corresponding EEPROM value [1])
Backlight intensity is set to the corresponding EEPROM value [1])
The backlight state is set to the corresponding EEPROM value [1])

Keyboard is enabled/disabled according to startup value
Keyboard notification is turn on/off according to startup value

All touch fields are removed, report touch field coordinates and movements is disabled

Relays are switched off

LED blink frequencies are set to the default values
LEDs are reset to power up state

Note:
If no corresponding value in EEPROM is available, the default values are taken from the setup values

Reset All and Show Startup Graphic

`<CID> $`

This command calls the "Reset All" (page 16) command and displays the startup graphics defined via the setup program.

Reboot Panel Controller

`<CID> #`

The panel controller is rebooted by preventing the internal watchdog from being serviced which gives the same effect as a hard reset at power up after sending the **<ACK>** character. This command is primarily useful after loading new data via the setup program (the setup program reboots the panel controller automatically after successfully loading the data). Please note that the baud rate for the serial port may change when the currently used baud rate is different from the value set via the setup program. The responding <ACK> is sent with the current baud rate, the # <ACK> sequence (see below) is sent with the new baud rate then.

When the controller gets to life again it returns # **<ACK>** to the default communications port if sending the startup messages is selected via the setup software.

Copyright © by demmel products, Document Date: December 2, 2009

Get Last Error Code

```
<CID> ? E
```

The controller returns <ACK> followed by a 2 byte error code and an <ACK> character. The error code is set by the last executed command. See the table Error Codes for a listing of all error codes.

Get Firmware Info

```
<CID> ? I
```

The controller returns <ACK> and the firmware info string followed by another <ACK> character.

Example:

```
<ACK>iLCD Firmware V1.0 (c) by demmel products 2003-2004<ACK>
```

Get Identification Info

```
<CID> ? M
```

The controller returns <ACK> and the identification info string followed by another <ACK> character.

Example:

```
<ACK>iLCD Firmware<ACK>
```

Get Firmware Version

```
<CID> ? V
```

The controller returns <ACK> and the text string "`x.yy`" where `x` and `y` describe the major and minor version number. The string is followed by another <ACK> character.

Example:

```
<ACK>1.00<ACK>
```

Get Serial Number

```
<CID> ? S
```

The controller returns <ACK> and a text string containing the unique serial number of the iLCD module. The string is followed by another <ACK> character.

Example:

```
<ACK> DH-XADC-00131154001-0566 <ACK>
```

Get iLCD Controller Name

```
<CID> ? C
```

The controller returns <ACK> and a text string containing the name of the iLCD controller of the module. The string is followed by another <ACK> character.

Example:

<ACK> DPC1030 <ACK>

Get Hardware Revision

```
<CID> ? H
```

The controller returns <ACK> and a text string containing the hardware revision of the iLCD controller of the module. The string is followed by another <ACK> character.

Example:

```
<ACK> 2.0 <ACK>
```

LCD-Control Commands

Please note that all display commands use a physical layout as follows:

The topmost pixel row is at y coordinate 0 and the leftmost pixel column is at x coordinate 0. Display page 0 describes the topmost 8 pixel rows and when writing to column 0 / page 0 with "Write Graphics Byte", a value of Hex 01, the top/left pixel is set.

Set Screen Orientation

```
<CID> C O orientation
```

Set the screen orientation where `orientation` is as follows:

0 Landscape mode (0°)
1 Portrait mode (90°)
2 Landscape mode topside down (180°)
3 Portrait mode topside down (270°)

Please note that switching the orientation causes a "Reset All" (see page 16) to be issued (without setting the default orientation). The default screen orientation can be set via the setup software.

En/disable ANSI

```
<CID> C A on_off
```

ANSI support is enabled when `on_off` is Hex 01 and disabled when `on_off` is hex 00. Please note that the default value is set via the setup program.

Set Page Address

```
<CID> C P address
```

`address` is a hex value between 0 and the maximum page available (height of display in pixels / 8 – 1). Please note that this command sets the current `y` position (see at "Set Pixel Coordinate" on page 20) to `address * 8` as well. Using values greater than the available pages causes a <NACK> to be sent.

Set Column Address

```
<CID> C C address_hb address_lb
```

`address` is a hex value between 0 and the display width - 1. Please note that setting column address also sets the `x` value of "Set Pixel Coordinate" on page 20. Using values greater than the available columns causes a <NACK> to be sent.

Increment/Decrement Column Address

```
<CID> C c addr_inc_hb addr_inc_lb
```

`addr_inc` is a hex value between *minus display width – 1* and *display width – 1* and specifies the amount of incrementing (positive value) or decrementing (negative value) the current cursor x-position in pixel units. Using an invalid increment/decrement value resulting in setting the cursor beyond the left or right margin of the display causes a <NACK> to be sent. Please note that setting column address also sets the `x` value of "Set Pixel Coordinate" on page 20.

Please see the chapter "Signed Words" on page 5 to learn about how to use negative values for `addr_inc`.

Set Row Address

```
<CID> C R address_hb address_lb
```

`address` is a hex value between 0 and the display height - 1. Please note that setting row address also sets the `y` value of "Set Pixel Coordinate" on page 20. Using values greater than the available rows causes a <NACK> to be sent.

Increment/Decrement Row Address

```
<CID> C r addr_inc_hb addr_inc_lb
```

`addr_inc` is a hex value between *minus display height – 1* and *display height – 1* and specifies the amount of incrementing (positive value) or decrementing (negative value) the current cursor y-position in pixel units. Using an invalid increment/decrement value resulting in setting the cursor beyond the top or bottom margin of the display causes a <NACK> to be sent. Please note that setting row address also sets the `y` value of "Set Pixel Coordinate" on page 20.

Please see the chapter "Signed Words" on page 5 to learn about how to use negative values for `addr_inc`.

Set Pixel Coordinate

```
<CID> C K x_hb x_lb y_hb y_lb
```

`x` is a hex value between 0 and display width – 1 and `y` a hex value between 0 and display height - 1. Set Pixel Coordinate is used in conjunction with Set/Clear Pixel, Draw Line, Draw Rectangle, Write Text and any other screen related command. Please note that this command sets the column address (see at "Set Column

Address") to `x` and the page address (see at "Set Page Address") to `y / 8.` Using invalid values for `x` or `y` causes a `<NACK>` to be sent.

Get Pixel Coordinate

```
<CID> C ? K
```

The controller returns `<ACK> x_hb x_lb y_hb y_lb <ACK>` where `x` is between 0 and display width - 1and `y` a hex value between 0 and display height – 1. The two values describe the actual pixel coordinate.

Get Text Extent

```
<CID> C ? T character1 character2 character3... null
```

As the panel controller also supports proportional fonts, it might be necessary to get information about the space a character string occupies on the LCD display. If you use a fixed pitch, non-symbolic font you could easily calculate the value by:

Number_of_horz_pixels = Number_of_characters x (current_font_width + 1)
Number_of_vert_pixels = current_font_height + 1

The panel controller returns `<ACK> x_hb x_lb y_hb y_lb <ACK>` to report the extent of the text given via `character1`, `character2 character3...` even for proportional fonts to allow centering text on the LCD display and so on.

Please note that any control characters like carriage returns and ANSI sequences are interpreted correctly if ANSI mode is enabled. Characters not defined in the current font table are calculated as full width spaces and this is how they are shown on the display. If you use a symbol font no horizontal or vertical spaces between the characters will be displayed, and the Get Text Extent command calculates its values accordingly. If text alignment is on (see "Set Text Alignment" on page 21), the reported height and width correspond to the setting of the alignment, however, the cursor position is not taken into account (that means no correction of the right and bottom margin will be made in this case).

Get Text Message Extent

```
<CID> C ? t index_hb index_lb
```

This command is the same as the "Get Text Extent" on page 21, but works with fixed message strings stored in the Flash memory of the iLCD via its setup program. If the message with the corresponding `index` is not available, a `<NACK>` is returned instead of the text extent string.

To get more information about fixed message strings, see the "Write Text Message" on page 30.

Set Text Alignment

```
<CID> C T mode widht_hb width_lb height_hb height_lb
```

This powerful command helps you to align text on the screen automatically. The following "Write Text" (see at page 30), "Write Text Message" (see at page 30) or "Get Text Extent" (see at page 21) commands align the word wrapped text corresponding to the `mode` set via this command. A maximum of 1024 characters and 48 text lines can be aligned.

`mode` consists of the following bits, which can be binary ORed together:

Bit 0 Center text horizontally
Bit 1 Center text vertically
Bit 2 Right justify text

Bit 3 Bottom justify text
Bit 4 Do not word wrap text
Bit 5 Add horizontal space for border
Bit 6 Add vertical space for border
Bit 7 Turn alignment on

Some bits do not make sense when used together, so there are precedent rules as follows:

Bit 0 takes precedence over bit 2 and bit 1 takes precedence over bit 3. This means that, when you specify horizontally center and right alignment, the text will be centered horizontally, and when you specify vertical center and bottom alignment, the text will be centered vertically.

Please note that bit 7 must be always set to enable alignment. This allows an accidentally set alignment to be cleared before the text is outputted.

The `width` and `height` value specify the "window" used for text alignment. If the `width` and/or the `height` parameter would exceed the resulting right/bottom margin when the command is executed (at the time when the "Write Text", "Write Text Message" or "Get Text Extent" command is executed), the controller automatically adjusts the `width` and/or `height` accordingly to the right margin. If the `width` and/or `height` parameter is set to zero, the controller automatically uses the maximum area available (starting from the current cursor position when the text output / get text extent command is issued).

After executing the "Write Text", "Write Text Message" or "Get Text Extent" command, the alignment is automatically cleared. For consecutive aligning of text strings, the set text alignment command has to be repeated.

Please note, that even ANSI command sequences (such as setting the font to bold or changing the font) can be used within the text to be aligned, although using cursor control ANSI commands within text alignments does not make sense and may produce unwanted results.

Carriage returns and linefeeds can be used to force a new line regardless of the actual horizontal space already used for the current line. Using carriage returns and linefeeds produce the same result in the case of outputting / getting the text extent of aligned text strings and the auto linefeed setting is ignored. Entering a CR/LF pair causes one new line, entering e.g. two consecutive CRs causes two new lines.

If the text does not fit into the given area, it will be truncated. If word wrap cannot be done due to words, which are longer than the available space, or word wrapping is switched off, the rest of the word will be continued on the next line.

Set Line Style

```
<CID> C L style
```

Lines can either have a solid style or appear e.g. dashed or dotted. `style` is a bit mask for line drawing where a 1 represents a pixel to be written and 0, a pixel to be omitted. This gives a value of hex FF for a solid line style and a value of hex 00 where the line is invisible. The least significant bit of style describes the first pixel painted when drawing a line from left to right.

Some examples for line styles:

Hex `FF`	(Binary `11111111`)	Solid line
Hex `55`	(Binary `10101010`)	Dotted line
Hex `F0`	(Binary `11110000`)	Dashed line (4 pixel black, 4 pixel white)
Hex `33`	(Binary `00110011`)	Dashed line (2 pixel black, 2 pixel white)
Hex `27`	(Binary `00100111`)	Dash-dotted line

At power up the line style is set to Hex FF.

Copyright © by demmel products, Document Date: December 2, 2009

Note that the actual line style is used when a rectangle is painted allowing the drawing of various styles of rectangles as well.

Get Display Size

`<CID> C ? D`

The panel controller returns `<ACK>` `width_hb width_lb height_hb height_lb` `<ACK>` to report the `width` and `height` of the LCD display. The actual display size is set via the setup program. The command is always acknowledged by an <ACK>.

Set TAB Spacing

`<CID> C S tab_spacing`

Set the tabulator spacing to `tab_spacing` (1 … 16). The next available multiple of this value is the next character position where the cursor is placed when a TAB character is outputted. Please note, that the font width (this is not the character width of single characters when using a proportional font) of the currently selected font is used for the cursor position calculation.

The power up value of `tab_spacing` is set via the setup program.

Set Auto-Linefeed

`<CID> C F on_off`

Set auto-linefeed to on (`on_off` = 1) or off (`on_off` = 0). When auto-linefeed is on, receiving a Hex 0D (carriage return) causes the cursor set to the left margin and a new line (line feed = Hex 0A) automatically added.

The power up value of auto-linefeed is set via the setup program.

Set Wrap Mode

`<CID> C W horz_wrap vert_wrap`

Set horizontal (`horz_wrap`) and vertical (`vert_wrap`) wrap mode to on (= 1) or off (= 0). When the horizontal wrap mode is on, characters outputted are automatically placed on a new line when there is not enough space on the current line anymore. When vertical wrap mode is on, the screen scrolls up when there is not enough space for the next line to be outputted to the LCD.

The power up values for `horz_wrap` and `vert_wrap` are set via the setup program.

Go Terminal Mode

`<CID> C G =`

This command allows the iLCD to leave the command mode and to enter the so-called terminal mode where the module acts like a standard ANSI terminal. After sending the `<ACK>` character, the only way to end the terminal mode is via the special escape sequence "End terminal mode" (see "Private ANSI extensions" on page 16). When the terminal mode is active, keystrokes are not reported via the make and break key anymore, but only the key code is reported. See chapter "The Concept of iLCD's Touch Fields" on page 42 and "Enable Keyboard Report" on page 50 for detailed information.

Issuing this command automatically enables the ANSI mode (see "En/disable ANSI" on page 19) and horizontal and vertical wrap mode (see "Set Wrap Mode" on page 23).

Copyright © by demmel products, Document Date: December 2, 2009

Please note that the terminal mode can be forced at power up when set via the setup program. Ending a terminal mode issued at startup is done via the same escape sequence as described above.

Set XON/XOFF For Terminal Mode

`<CID> C X on_off`

When the module is set to the terminal mode, the previously set XON/OFF mode becomes active. If XON/XOFF is on (`on_off` = 1) the iLCD module sends an XOFF character (Hex 13) when the input buffer (the size of input buffer is 128 characters) gets filled with 64 characters ("high-water") and sends an XON character (Hex 11) when the buffer is emptied to $^1/_3$ ("low water") of the input buffer size. This is a standard mechanism used for terminals which do not have the ability to use hardware handshake signals (see some general information about this issue on "Command Structure" on page 1 and "Terminal Mode" on page 3.

Please note that the XON/XOFF mode is not active when the module is in standard command mode.

The power up value for XON/XOFF mode is set via the setup program.

Set Backlight Mode

`<CID> C B mode`

Set backlight to `mode`, where `mode` is as follows:

0 = Backlight Off
1 = Backlight On
2 = Blink Backlight
3 = Fade Out Backlight (only when previously on or blinking)

The backlight state at power up is retrieved from the internal EEPROM section of the controller (location 2). When the EEPROM is erased the default power up value is got from the Flash data set via the setup program. See further information about EEPROM at section "EEPROM Related Commands" on page 55.

Get Backlight Mode

`<CID> C ? B`

This command allows the application to retrieve the current state of the backlight. The iLCD module responds with `<ACK> mode <ACK>` where `mode` is as follows:

0 = Backlight is off
1 = Backlight is on
2 = Backlight is blinking

Set Backlight Blink Frequency

`<CID> C b frequency`

`frequency` is a value between 1 and Hex FF and describes the number of 10 ms intervals until the state changes from on to off and *vice versa*. So, for example, a value of decimal 25 (Hex 19) gives a frequency of 2 Hertz (250ms on and 250ms off). The default blink frequency at startup can be defined via the setup program. When the backlight is currently in blink mode, while the command is issued, the blink frequency changes automatically.

Set Backlight Intensity

```
<CID> C I intensity
```

Setting the `intensity` of the LCD backlight can be done in 16 steps from 0 to 15 where 15 indicates full brightness and 0 the minimum brightness (that means backlight is still on).

The backlight intensity at power up is retrieved from the internal EEPROM section of the controller (location 1). When the EEPROM is erased, the default power up value is gotten from the Flash data set via the setup program. See further information about EEPROM at section "EEPROM Related Commands" on page 55. Switching the backlight on or off is done via a separate command (see at "Get Backlight Mode" on page 25), and setting the backlight intensity does not turn the backlight on or off.

Get Backlight Intensity

```
<CID> C ? I
```

This command allows the application to retrieve the current backlight intensity (even when the backlight is actually off). The iLCD module responds with `<ACK> intensity <ACK>` where `intensity` is a value between 0 and 15 (see "Set Backlight Intensity" on page 25). The command is always acknowledged by an <ACK>.

Get Fixed LCD Contrast/Gamma

Some newer color iLCD panels do not need to have the contrast and gamma value set, as they have the optimum values already set by the TFT panel manufacturer. These values cannot be modified in that case. Although setting and reading the contrast via the appropriate commands can be carried out, there is simply no visual effect on setting the values.

To find out if a certain iLCD panel has a fixed (= not adjustable) contrast and gamma value, issue the command as follows:

```
<CID> C ? X
```

The response to this command is as follows:

```
<ACK> fixed <ACK>
```

If the contrast and the gamma value can be adjusted, `fixed` is returned as 0, else `fixed` is returned as 1.

Please note, that this command is not support on firmware versions < 1.19, so when getting a `<NACK>` sent instead of the above mentioned sequence, one can assume for sure, that gamma and contrast values can be adjusted. In this case the error code may be requested by the Get Last Error Code Command.

Set LCD Contrast

```
<CID> C N value
```

`value` is a value between 0 and Hex FF and describes the contrast of the LCD display.

The contrast at power up is retrieved from the internal EEPROM section of the controller (location 0). When the EEPROM is erased the default power up value is got from the Flash data set via the setup program. See further information about EEPROM at section "EEPROM Related Commands" on page 55.

Get LCD Contrast

```
<CID> C ? N
```

This command allows the application to retrieve the current LCD contrast. The iLCD module responds with `<ACK> contrast <ACK>` where `contrast` is a value between 0 and 255 (Hex FF). The command is always acknowledged by an <ACK>.

Set LCD Gamma Value

```
<CID> C M value
```

`value` is a value between 0 and Hex FF and describes the gamma value of the LCD display.

The gamma value at power up is retrieved from the internal EEPROM section of the controller (location 3). When the EEPROM is erased the default power up value is got from the Flash data set via the setup program. See further information about EEPROM at section "EEPROM Related Commands" on page 55.

Get LCD Gamma Value

```
<CID> C ? M
```

This command allows the application to retrieve the current LCD gamma value. The iLCD module responds with `<ACK> gamma <ACK>` where `gamma` is a value between 0 and 255 (Hex FF). For displays for which this command is not supported, a <NACK> is returned in this case.

LCD Attribute Commands

Set Font

```
<CID> A F number_hb number_lb
```

`number` is the index of the font to select. `number` can be between hex 0 and the number of available fonts - 1. Font 0 is always available (even when the Flash is blank).

Please note that setting a font also clears the underline position (see "Set Underline Position" on page 27) to 0 and sets the font spacing (see "Set Font Spacing" on page 26) and symbol font mode (see "Set Symbol Font" on page 27) to the way the font was designed via the setup program.

Please note that there is a private ANSI extension too, which allows setting the font via an escape sequence (see "Private ANSI extensions" on page 16).

Set Font Spacing

```
<CID> A S x_spacing y_spacing
```

This command allows you to overwrite the font spacing values defined via the setup program for the currently selected font. `x_spacing` describes the number of blank pixels between two consecutive characters in horizontal direction and `y_spacing` describes the number of blank pixels between two consecutive lines. `x_spacing` and `y_spacing` can have a value between 0 and decimal 15.

If the current font is set to symbol mode (see "Set Symbol Font" on page 27) the setting of `x_spacing` and `y_spacing` is ignored and no blank space is used between consecutive characters and lines.

Copyright © by demmel products, Document Date: December 2, 2009

Set Symbol Font

`<CID> A Y on_off`

If `on_off` is 1 the currently selected font is drawn without any blank space between consecutive characters and lines. This is especially useful when using fonts which contain border characters. Please note that selecting a font which was defined as a symbol font via the setup program does not require the issue of this command. If `on_off` is 0, symbol mode is turned off and the default font spacing is used or when previously overwritten via the "Set Font Spacing" (page 26), the last selected font spacing is used.

Set Bold Mode

`<CID> A B on_off`

Any font can be drawn in bold mode by adding an extra pixel row to the characters to be printed. This is done automatically by the iLCD controller when `on_off` is set to 1. When bold mode is activated, all characters occupy one more pixel on the display in the horizontal direction. When using the appropriate alignment functions, this is taken into account automatically, but when running the terminal mode and using cursor control ANSI sequences, this may produce unwanted results (see at Terminal Mode on page 3).

Setting `on_off` to 0 clears the bold mode accordingly.

Please note that there is also an ANSI escape sequence (see "Attribute Sequences" on page 15) to enable and disable bold mode.

Set Underline Mode

`<CID> A U on_off`

When turning underline mode on (`on_off` = 1), any character is underlined. This means that a line is drawn at the lowest pixel position, which is within the character area. The vertical position of the underlining can be set via the "Set Underline Position" on page 27.

Set Underline Position

`<CID> A u position`

`position` is a signed byte value (see at "Signed Bytes" on page 5), can have a value between +31 and –32 and describes the vertical position of underlining. If the current vertical font spacing is 1 (see at "Set Font Spacing" on page 26), any value greater than 0 for `position` causes the underline not drawn anymore as it would overwrite parts of next line. Using a negative value for `position` will cause the line to be drawn within the character area (thus crossing out instead of underlining the character). If `position` would cause overwriting parts of the previous line (depending on the font height), underlining is not carried out anymore.

Please note that setting a font (see at "Set Font" on page 26) causes `position` automatically to be reset to 0.

Set Inverse Mode

`<CID> A I on_off`

If `on_off` is hex 0, inverse mode is off; otherwise inverse mode is set to on. All text and graphic output commands take care of the inverse mode flag (for example, Write Text, Write Graphics Byte, Erase Area etc.)

At startup inverse mode is off.

Set Transparent Mode On/Off

```
<CID> A T on_off
```

Normally character output wipes out the space (character_width + 1 * character_height * 1 for non symbol fonts – depending on the currently selected font spacing) where the character is displayed before drawing the character. If you want to draw text into a graphic already existing on the LCD without wiping out the background, you may turn the transparent mode on by setting `on_off` to 1. At startup the transparent mode is off.

Set Foreground Color

```
<CID> A C F red green blue
```

Sets the actual foreground color characters and lines are drawn with. At startup and after "Reset All" the foreground color is set to black.

The 24-bit color value consists of the sub-parts `red`, `green` and `blue` according to "24-Bit Color Values" on page 5.

Please note, that monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.

Set Background Color

```
<CID> A C B red green blue
```

Sets the actual background color. At startup and after "Reset All" the foreground color is set to black.

The 24-bit color value consists of the sub-parts `red`, `green` and `blue` according to "24-Bit Color Values" on page 5.

If the transparent mode (see at "Set Transparent Mode On/Off" on page 28) is not off, the character's background is drawn with the current background color. Using the "Erase Display" (see page 29) or the "Erase Display Area" (see page 29) erases the display/area with the current background color.

Please note, that monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.

Set Border Color

```
<CID> A C R red green blue
```

Sets the actual border color. At startup and after "Reset All" the border color is set to black.

The 24-bit color value consists of the sub-parts `red`, `green` and `blue` according to "24-Bit Color Values" on page 5.

Rectangles drawn with the "Draw Rectangle" command (see page 34) are drawn with the current border color. If the rectangle has shadows, these shadows are drawn with the current border shadow color according to "Set Border Shadow Color" (see page 29).

Set Border Shadow Color [3]

```
<CID> A C S red green blue
```

Sets the actual border shadow color. At startup and after "Reset All" the border color is set to grey.

The 24-bit color value consists of the sub-parts `red`, `green` and `blue` according to "24-Bit Color Values [3]" on page 5.

Rectangles drawn with the "Draw Rectangle" command (see page 34) are drawn with the current border color according to Set Border Color [3] on page 28. If the rectangle has shadows, these shadows are drawn with the current border shadow color.

LCD Draw Commands

Erase Display

```
<CID> D E
```

[1][2] Clears the entire display. If inverse mode is set all pixels are set when the clear command is processed, otherwise all pixels are cleared. The command is always acknowledged by an <ACK>.

[3] Clears the entire display to the current background color. If inverse mode is set to true, the display is cleared to the foreground color instead.

Erase Display Area

```
<CID> D e width_hb width_lb height_hb height_lb
```

Erases an area with size `width` and `height` beginning from the current cursor position x, y. If the current cursor position plus the `width` / `height` exceeds the right / bottom margin, the iLCD controller corrects the values accordingly without notifying an error. The maximum value for `width` and `height` are determined by the display width and height.

[1][2] If inverse mode is set, all pixels within the given area are set when the clear command is processed otherwise all pixels are cleared.

[3] If inverse mode is off, the area is cleared to the background color; otherwise the foreground color is used.

Invert Screen

```
<CID> D I
```

[1][2] All black pixels on the screen are set to blank and vice versa.
[3] Any pixel on the screen gets the color value inverted, e.g. black pixels become white and red pixels become cyan and vice versa. The command is always acknowledged by an <ACK>.

Using this command does not turn the inverse mode (see "Set Inverse Mode" on page 27) on or off.

Invert Screen Area

```
<CID> D i width_hb width_lb height_hb height_lb
```

[1][2] All black pixels within the area of `width` and `height` – beginning from the current cursor position - are set to blank and vice versa.

All pixels within the area of `width` and `height` – beginning from the current cursor position - are set to the inverted color, e.g. black pixels become white and red pixels become cyan and vice versa.

If the current cursor position plus the `width` / `height` exceeds the right / bottom margin, the iLCD controller corrects the values accordingly without notifying an error. The maximum value for `width` and `height` are determined by the display width and height.

Using this command does not turn the inverse mode (see "Set Inverse Mode" on page 27) on or off.

Write Text

```
<CID> D T character1 character2 character3... null
```

Write a text to the current cursor position. `character` can be a value between Hex 01 and Hex FF (including); characters not defined in the character table are replaced by a space. A `NULL` character terminates the sequence. If the column address exceeds the right or bottom margin during printing and the corresponding wrap mode is not set, the following characters are ignored. Please note that outputting text also increases the column address accordingly, so more graphics or text drawn after outputting text will appear after the last text character. To get the horizontal and vertical space occupied by the outputted text, use the Get Text Extent command below. The text can contain any ANSI sequences (see at "ANSI Support" on page 14) to e.g. control the cursor position (on a character OR graphic based position) when the ANSI mode is on.

When an align mode was previously set (see the "Set Text Alignment" command on page 21), text aligning will be carried out accordingly and the align mode will be cleared after the text has been outputted.

Write Text Message

```
<CID> D t index_hb index_lb
```

This command works in the same way as the "Write Text" command on page 30, but a previously set text (via the setup program) addressed via `index` will be outputted. Please note that these "fixed" text strings can contain ANSI sequences as well.

Scroll Up Screen

```
<CID> D S U scroll_y_hb scroll_y_lb
```

Scrolls up the screen contents by `scroll_y` pixels. The maximum value for `scroll_y` is determined by display height - 1.

Scroll Down Screen

```
<CID> D S D scroll_y_hb scroll_y_lb
```

Scrolls down the screen contents by `scroll_y` pixels. The maximum value for `scroll_y` is determined by display height - 1.

Scroll Left Screen

```
<CID> D S L scroll_x_hb scroll_x_lb
```

Scrolls left the screen contents by `scroll_x` pixels. The maximum value for `scroll_x` is determined by display width - 1.

Scroll Right Screen

```
<CID> D S R scroll_x_hb scroll_x_lb
```

Scrolls right the screen contents by scroll_x pixels. The maximum value for scroll_x is determined by display width - 1.

Read Graphics Byte 1 2

```
<CID> D ? R
```

The controller acknowledges the command and reads one byte from the current column/page address and increments the column address afterwards. Please note that no character quoting is done on up streaming data. The byte to be read is returned after the <ACK> to the command in hex followed by the usual <ACK> character (Hex 06) character. If the byte to be read has the value Hex 06 the response will be Hex 06, Hex 06, Hex 06.

Read Multiple Graphics Byte 1 2

```
<CID> D ? r count_hb count_lb
```

The iLCD controller reads count bytes from the current column/page address and increments the column address after every read. If count + the actual column value exceed the right margin, an error is reported instead of reporting the data.

Write Graphics Byte 1 2

```
<CID> D W N byte
```

Write a single byte into the LCD controller graphics RAM. If the byte to be written is the <CID> character, a preceding <CID> character must quote it. The location where the byte is written to is determined by the current column/page address. The column address is incremented after the byte has been written.

Write Multiple Graphics Bytes 1 2

```
<CID> D w N count_hb count_lb bytes
```

Write count (hex 1 .. display width) bytes into the LCD controller graphics RAM. If a byte to be written is the <CID> character, it must be quoted by a preceding <CID> character. Please note that count describes the "raw" data bytes only, which means a quoted <CID> character counts as one character. The location where the byte is written to is determined by the current column/page address. The column address is incremented after every byte of the stream has been written. If the column address exceeds the right margin during printing, <NACK> is returned and the command processing returns and waits for the next <CID> character.

Binary OR Graphics Byte 1 2

```
<CID> D W O byte
```

Write a single byte into the LCD controller graphics RAM by first reading the current location, ORing byte to this value and writing it back to the previous position. If the byte to be written is the <CID> character, a preceding <CID> character must quote it. The location where the byte is written to is determined by the current column/page address. The column address is incremented after the byte has been written.

Binary OR Multiple Graphics Bytes 1 2

```
<CID> D w O count_hb count_lb bytes
```

Write **count** (hex 1 .. display width) bytes into the LCD controller graphics RAM by first reading the current location, ORing **byte** to this value, and writing it back to the previous position. If a byte to be written is the **<CID>** character, it must be quoted by a preceding **<CID>** character. Please note that **count** describes the "raw" data bytes only, which means that a quoted **<CID>** character counts as one character. The location where the byte is written to is determined by the current column/page address. The column address is incremented after every byte of the stream has been written. If the column address exceeds the right margin during printing, **<NACK>** is returned and the command processing returns and waits for the next **<CID>** character.

Binary AND Graphics Byte 1 2

```
<CID> D W A byte
```

Write a single byte into the LCD controller graphics RAM by first reading the current location, ANDing **byte** to this value, and writing it back to the previous position. If the byte to be written is the **<CID>** character, a preceding <CID> character must quote it. The location where the byte is written to is determined by the current column/page address. The column address is incremented after the byte has been written.

Binary AND Multiple Graphics Bytes 1 2

```
<CID> D w O count_hb count_lb bytes
```

Write **count** (hex 1 .. display width) bytes into the LCD controller graphics RAM by first reading the current location, ANDing **byte** to this value, and writing it back to the previous position. If a byte to be written is the **<CID>** character, it must be quoted by a preceding **<CID>** character. Please note that **count** describes the "raw" data bytes only, which means that a quoted **<CID>** character counts as one character. The location where the byte is written to is determined by the current column/page address. The column address is incremented after every byte of the stream has been written. If the column address exceeds the right margin during printing, **<NACK>** is returned and the command processing returns and waits for the next **<CID>** character.

Binary XOR Graphics Byte 1 2

```
<CID> D W X byte
```

Write a single byte into the LCD controller graphics RAM by first reading the current location, XORing **byte** to this value and writing it back to the previous position. If the byte to be written is the **<CID>** character, a <CID> character must quote it. The location where the byte is written to is determined by the current column/page address. The column address is incremented after the byte has been written.

Binary XOR Multiple Graphics Bytes 1 2

```
<CID> D w X count_hb count_lb byte1 byte2 ...
```

Write **count** (hex 1 .. display width) bytes into the LCD controller graphics RAM by first reading the current location, XORing **byte** to this value and writing it back to the previous position. If a byte to be written is the **<CID>** character, it must be quoted by a preceding **<CID>** character. Please note that **count** describes the "raw" data bytes only, which means that a quoted **<CID>** character counts as one character. The location where the byte is written to is determined by the current column/page address. The column address is incremented after every byte of the stream has been written. If the column address exceeds the right margin

Copyright © by demmel products, Document Date: December 2, 2009

during printing, `<NACK>` is returned and the command processing returns and waits for the next `<CID>` character.

Write Scan Line 3

```
<CID> D N W no_of_pixels_hb no_of_pixels_lb p0_hb p0_lb p1_hb p1_lb ...
```

Write a (partial) horizontal scan line consisting of `no_of_pixels_hb` pixels (hex 1 .. display width) into the LCD controller graphics RAM. Data for the single pixels `p0`, `p1`, ... is made up as a 16-bit color value according to 16-Bit Color Values 3 on page 4.

Any byte after the leading `<CID>` character being a `<CID>` character (e.g. a color value containing the Hex byte AA) must be quoted by a preceding `<CID>` character.

The location where the pixel is written to is determined by the current column/row address. The column address is incremented after every pixel. If the column address exceeds the right margin during printing, `<NACK>` is returned and the command processing returns and waits for the next `<CID>` character.

Read Scan Line 3

```
<CID> D N R no_of_pixels_hb no_of_pixels_lb
```

The iLCD controller reads `no_of_pixels` pixels from the current column/row address and increments the column address after every read. If `no_of_pixels` + the actual column value exceed the right margin, an error is reported instead of reporting the data.

The response to this command consists of an `<ACK>` followed by the pixel data read from the display which are made up as a data stream of `no_of_pixels` 16-bit color values according to 16-Bit Color Values 3 on page 4 followed by an terminating `<ACK>`

Set/Clear Pixel

```
<CID> D P on_off
```

Sets / clears the pixel at the current location. If `on_off` is `1`, the pixel is set, if `on_off` is `0`, the pixel is cleared. No advancing of the current X/Y position will occur.

1 2 If inverse mode is set to true, setting/clearing a pixel will be done in the opposite way.
3 Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.

Set/Clear Pixel At X/Y

```
<CID> D p x_pos_hb x_pos_lb y_pos_hb y_pos_lb on_off
```

Sets / clears the pixel at the location `x_pos/y_pos`. If `on_off` is `1`, the pixel is set, if `on_off` is `0`, the pixel is cleared. The current x/y position used by e.g. "Draw Line" is set accordingly.

1 2 If inverse mode is set to true, setting/clearing a pixel will be done in the opposite way.
3 Setting/clearing a pixel is done with the current foreground/background color. When inverse mode is on, setting/clearing a pixel will be done in the opposite way.

Draw Line

```
<CID> D L end_x_hb end_x_lb end_y_hb end_y_lb
```

Draws a line with the currently selected line style (see at "Set Line Style" on page 22) from the current x/y location to `end_x/end_y`. Please note that the point `end_x/end_y` is included and the current x/y position is set to `end_x/end_y` after the line has been drawn.

1 2 If inverse mode is set to true, drawing a line will be done with white pixels instead of black pixels.
3 Drawing a line is done with the current foreground color. When inverse mode is on, the current background color is used instead.

Draw Rectangle

```
<CID> D R mode width_hb width_lb height_hb height_lb
```

Draws a rectangle with the currently selected line style (see at "Set Line Style" on page 22) from the current x/y location with a size of `width/height`. Please note that the current x/y position is reset to the starting coordinates after the rectangle has been drawn.

`mode` is a value consisting of the following bits:

Bit 0	The rectangle is drawn with rounded corners when set
Bit 1	When set, a shadow is drawn at the right and the bottom part of the rectangle
Bit 2	Blank pixels are drawn outside of the rectangle if set (to distinguish from the background)
Bit 3	The inner part of the rectangle will be erased when set
Bit 4	Erasing of the inner part is done with the inverse setting (pixel set instead of cleared)

All drawings depend on the global inverse flag, so e.g. erasing the inner part will draw black pixels if the inverse mode is on.

3 Drawing the rectangle is done with the current border color; the shadow is drawn with the current border shadow color. When erasing the inner part of the rectangle, the current background color is used. When inverse mode is on, all colors are inverted accordingly.

Please note that when drawing a shadow or drawing blank pixels outside of the rectangle, drawing will be done outside of the area selected via `width` and `height`.

Draw Circle

```
<CID> D C radius_hb radius_lb
```

A circle with radius (maximum value is determined by the display width) with the center point of the current X/Y coordinate is drawn.

1 2 If inverse mode is set to true, drawing a circle will be done with white pixels instead of black pixels.
3 Drawing a circle is done with the current foreground color. When inverse mode is on, the current background color is used instead.

LCD Graphics Commands

Display Local Graphic

```
<CID> G graphic_index_hb graphic_index_lb
```

Paint the locally stored graphic (loaded via the setup program) with index number `graphic_index` at the current X/Y position. Column and row address are not changed. Please note that the inverse mode is not changed, which means that when inverse mode is on, the graphic is shown in inverse mode. If the graphic with graphic_index is not available, <NACK> is returned.

If graphic_index refers to an animated graphic, only the first frame of the graphic is painted.

Please note, that monochrome graphics drawn on a color iLCD panel have the originally black pixels shown in the current foreground value and the originally white pixels shown in the current background color allowing to "dye" monochrome graphics.

General Information About Animated Graphics

All graphics – including the animated ones – are loaded via the setup program into the Flash memory of the iLCD controller. Although there can be an unlimited number of animated graphics (limited by the available Flash memory size only), a maximum of 8 animated graphics can be animated at the same time on a LCD screen. While animated graphics are shown on the screen, any other command like drawing a text or a graphic to any cursor position can be carried out, as the iLCD's animation engine runs in the background of the iLCD controller firmware. Scrolling the screen while animations are active can be done, but will cause unwanted effects, as the position of the animated graphics are not changed via the scroll screen commands.

When working with animated graphics, the first thing to do is to load an animated graphic to one of the 8 *animation controls* (referred as anim_loc in the following commands), as all other animation related commands refer to this animation control regardless of which animated graphic is loaded on this control.

Please note that the single frames of an animated graphic may contain areas of the image, which are smaller in width and/or height than the complete image. This is exactly how animated GIFs work (the setup program imports animated GIFs and takes care about the frames' size and position), so setting a certain frame number via the "Stop And Set Frame Number" command (page 36) may cause surprising effects when one does not keep in mind, that only the (smaller) frame contents of the selected frame will be drawn, and not the whole image which is normally created by the sequence of consecutive frames.

Load Animated Graphics

```
<CID> g L anim_loc index_hb index_lb
```

Load an animated graphic to the animation control with index anim_loc (0 … 7). If the index of the graphic does not exist or the graphic to be loaded is not an animated graphic, a <NACK> is returned. Note that issuing this command does not show or start the animated graphic yet, it only assigns the graphic to the animation control.

Set Coordinates To X, Y

```
<CID> g K anim_loc pos_x_hb pos_x_lb pos_y_hb pos_y_lb
```

When an animated graphics is loaded via the "Load Animated Graphics" command (page 35), the initial coordinate is set to the current x/y cursor address. Using this command allows you to set the position where the animated graphics will be drawn (although this command does not paint anything). Setting the coordinate without stopping an already running animation may cause unwanted effects. If anim_loc has never been loaded with an animated graphic before or pos_x and/or pos_y exceed the display width/height, a <NACK> is returned.

Set Coordinates To Current Screen Coordinates

```
<CID> g k anim_loc
```

Using this command allows you to set the position where the animated graphics will be drawn (although this command does not paint anything) to the current screen coordinate. Setting the coordinate without stopping an already running animation may cause unwanted effects. If `anim_loc` has never been loaded with an animated graphic before a <NACK> is returned.

Start Or Restart Animation

```
<CID> g S anim_loc
```

This command starts or restarts (if previously stopped) the animated graphic loaded to the animation control referred by `anim_loc`. If the animated graphic does not run endlessly (that means the number of repetitions are greater than 0), the internal repetition counter is reset before the animation is (re)started. If `anim_loc` has never been loaded with an animated graphic before, a <NACK> is returned. If the animation or animation control `anim_loc` is already running, this command has no effect.

Stop And Set Frame Number

```
<CID> g F anim_loc frame_hb frame_lb
```

If the animation control referred by `anim_loc` currently animates a graphic, the animation is stopped. The animated graphics frame with index `frame` (0 … number of frames – 1) is painted on the location set via the "Set Coordinates To X, Y" command (page 35). If `frame` does not exist or `anim_loc` has never been loaded with an animated graphic before, a <NACK> is returned.

Stop (Break) Animation

```
<CID> g B anim_loc
```

Stops the animation or animation control anim_loc. If `anim_loc` has never been loaded with an animated graphics before, a <NACK> is returned. If the animation or animation control `anim_loc` is not running, this command has no effect. Animations stopped via this command can be restarted via the "Start Or Restart Animation" command (page 36) exactly where they have been stopped.

Stop (Break) All Animations

```
<CID> g B A
```

This command stops all animations currently active. The single animations can then be restarted via the "Start Or Restart Animation" command (page 36).

Set Repetitions

```
<CID> g R anim_loc repeat_hb repeat_lb
```

Normally the repetitions of an animated graphic are set via the setup program, but the number of repetitions until the animation stops automatically can be overwritten via this command. A `repeat` value of 0 sets unlimited repetitions, a value of e.g. 3 will set the graphic to be animated 3 times when started with the "Start Or Restart Animation" command on page 36. If `anim_loc` has never been loaded with an animated graphics before, a <NACK> is returned.

Erase Image Area

```
<CID> g E anim_loc
```

This command erases the area covered by the complete animated image. If `anim_loc` has never been loaded with an animated graphics before, a <NACK> is returned.

Erase Frame Area

```
<CID> g e anim_loc
```

This command erases the area covered by the current frame of the animated image. Note that the area may be smaller than the area of the complete image and that the upper left corner of the area to be erased is given by the X/Y offset of the frame. If `anim_loc` has never been loaded with an animated graphics before, a `<NACK>` is returned.

Suspend Animation Engine

```
<CID> g s
```

This command allows for the stopping of all animated graphics by stopping the animation engine without having to restart the single animations afterwards. Use the "Resume Animation Engine" command (page 37) to restart all animated graphics at the same point where they were stopped before.

Using this command is highly appreciated before reading the LCD's screen contents via the "Read Multiple Graphics Byte 1 2" (page 31), as scrambled screen contents may be read when animations are active.

Resume Animation Engine

```
<CID> g r
```

Resume the animation engine previously stopped via the "Suspend Animation Engine" command (page 37).

Screen Memory Related Commands

The iLCD controller contains some RAM for storing complete screen contents. This allows the user to store the current screen, paint some things (e.g. a box with a query) above an area of the screen and then restore the previous screen instead of rebuilding the previously screen’s content. The number of memory positions is dependent on the width and height of the LCD currently in use and the iLCD controller type itself (there are models with larger RAM). When using a 128 x 64 pixel LCD, six user accessible memory positions are available. Using a smaller or larger LCD changes the number of screen memory positions accordingly; the number of memory positions is limited to a maximum of eight.

3 Color iLCDs need a rather high amount of memory to store a screen (width * height * 2 bytes), so these models use an external RAM of typical 1 MByte size to store screens into. Depending on the model this RAM may not be included in your board, retrieving the number of screen memory positions with "Get # Of Screen Memory Positions" (see page 37) returns 0 in this case.

Please note that all screen memory related commands (except "Recall Screen From Memory" on page 38and "Save Screen To Memory" on page 38) take care of the currently set inverse mode (see "Set Inverse Mode" on page 27) that means that, for example, scrolling up a stored screen causes black pixels (or foreground-colored pixels when using a color iLCD) at the bottom of the screen image when the inverse mode is set to true.

Get # Of Screen Memory Positions

```
<CID> M S ?
```

Before calling any of the below mentioned commands, the number of available screen memory positions should be retrieved. The controller returns `<ACK> number <ACK>` where `number` describes the number of available screen memory positions. Please note that this number may be zero when using large LCDs and, if so, the screen memory related commands cannot be used then. The maximum available screen memory positions are limited to 8 regardless of the controller RAM available. The command is always acknowledged by an <ACK>.

Save Screen To Memory

<CID> M S S index

Capture the current screen contents to screen memory position index (0 … number of screen memory positions – 1).

If index exceeds the available screen memory positions, <NACK> is returned. In this case the error code may be retrieved by the Get Last Error Code Command.

Recall Screen From Memory

<CID> M S C index

Restore a previously captured screen's content to the LCD. index (0 … number of screen memory positions – 1) describes the screen memory position. The currently set inverse mode (see "Set Inverse Mode" on page 27) is ignored (different to "Paint Stored Screen" on page 38).

If index exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Paint Stored Screen

<CID> M S P index

Paint the image contained in screen memory position index to the LCD at the current X/Y position where index (0 … number of screen memory positions – 1) describes the screen memory position. The currently set inverse mode (see "Set Inverse Mode" on page 27) is taken into account accordingly.

If index exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Invert Stored Screen

<CID> M S I index

Invert the contents of the screen memory position addressed by index.

If index exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Scroll Up Stored Screen

<CID> M S U index scroll_y_hb scroll_y_lb

Scroll up the contents of the screen memory position addressed by index by scroll_y pixels. The currently set inverse mode (see "Set Inverse Mode" on page 27) is taken into account accordingly. The maximum value for scroll_y is determined by display height - 1.

If index exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Scroll Down Stored Screen

`<CID> M S D index scroll_y_hb scroll_y_lb`

Scroll down the contents of the screen memory position addressed by `index` by `scroll_y` pixels. The currently set inverse mode (see "Set Inverse Mode" on page 27) is taken into account accordingly. The maximum value for `scroll_y` is determined by display height - 1.

If `index` exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Scroll Left Stored Screen

`<CID> M S L index scroll_x_hb scroll_x_lb`

Scroll left the contents of the screen memory position addressed by `index` by `scroll_x` pixels. The currently set inverse mode (see "Set Inverse Mode" on page 27) is taken into account accordingly. The maximum value for `scroll_x` is determined by display width - 1.

If `index` exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Scroll Right Stored Screen

`<CID> M S R index scroll_x_hb scroll_x_lb`

Scroll right the contents of the screen memory position addressed by `index` by `scroll_x` pixels. The currently set inverse mode (see "Set Inverse Mode" on page 27) is taken into account accordingly. The maximum value for `scroll_x` is determined by display width - 1.

If `index` exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Set Height Of Stored Screen

`<CID> M S H index height_hb height_lb`

Set the `height` of the screen memory position addressed by `index`. If the height of an image is decreased the bottom part of the image will be cut. The maximum value for `height_x` is determined by the display height.

If `index` exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Set Width Of Stored Screen

`<CID> M S W index width_hb width_lb`

Set the `width` of the screen memory position addressed by `index`. If the width of an image is decreased the right hand part of the image will be cut. The maximum value for `width_x` is determined by the display width.

If `index` exceeds the available screen memory positions or no screen has been captured to the screen memory position before, <NACK> is returned.

Cursor Memory Related Commands

The iLCD controller contains 8 memory locations for saving the cursor position, the current attributes (see "Set Bold Mode" on page 27, "Set Underline Mode" on page 27, "Set Inverse Mode" on page 27 and "Set Transparent Mode On/Off" on page 28) and the currently selected font.

Color iLCD panels save all color values (background, foreground, border and border shadow color) as well.

This enables the user to easily save and restore the actual state of commonly used screen locations when outputting text to the display.

Save Cursor & Attributes To Memory

`<CID> M C S index`

Save cursor position, attributes and font index to position `index`. `index` can have a value between 0 and 7. See the general information about cursor memory at "Cursor Memory Related Commands" on page 40.

Color iLCD panels save all color values (background, foreground, border and border shadow color) as well.

Restore Cursor & Attributes from Memory

`<CID> M C C index`

Restore cursor position, attributes and font index to position `index`. `index` can have a value between 0 and 7. If no previous saving has been done to this index, `<NACK>` is returned. See the general information about cursor memory at "Cursor Memory Related Commands" on page 40.

Color iLCD panels save all color values (background, foreground, border and border shadow color) as well.

Macro Related Commands

Macros are a very powerful feature of the iLCD panel controllers and allow you to condense multiple commands into one macro which can be executed at any time via a command.

Macros can even call other macros with the the maximum nesting depth of 8. If the nesting depth is exceeded, the macro will be stopped and a `<NACK>` is returned to the user. The same is true if one of the commands executed returns an error. If all commands within the macro are executed successfully, one `<ACK>` is sent back to the user. This means that not all command responses within a macro are seen by the user.

If a macro is active and a `<CID>` character is entered, the currently running macro is stopped. No response is sent in this case as the command following the `<CID>` has to be acknowledged. Execution of macros will also be stopped if a key is pressed and key reporting is enabled, or when the ATX power switch is pressed and power-off notification is on.

To protect macros from being aborted, a macro timer can be set. For the defined time, all incoming commands are then kept in the input buffer. These commands are executed as soon as either the macro has finished running or the macro timer runs out. In the latter case, the currently running macro is stopped.

Macros are stored in the iLCD controllers Flash via the setup program and can be tested via the terminal mode of the setup program, although macro delays and execution speed settings are ignored in normal command mode.

Copyright © by demmel products, Document Date: December 2, 2009

Execute Macro

`<CID> O E index_hb index_lb`

Execute (call) macro referred via `index`. If the macro does not exist or one of the called commands returns an error, `<NACK>` is returned. When the macro is fully executed, an `<ACK>` is sent. Macros can call other macros, as long as the maximum nesting level is not exhausted. If a macro is executed within another macro ("nested"), the calling macro is resumed after the position where the macro was called once the called macro is finished.

For building endless loops, uses the "Jump to Macro" on page 41 to jump to the beginning of the current macro instead of using the "Execute Macro" command (this would overflow the macro stack and an error would be reported).

Jump to Macro

`<CID> O J index_hb index_lb`

Execute the macro referred by `index`. If the macro does not exist or one of the called commands returns an error, `<NACK>` is returned. Using "Jump to Macro" only makes sense at the end of a macro to branch to, say, the beginning of the current macro for building endless loops, as this command does not use the macro stack.

Delay Macro Execution

`<CID> O D delay_hb delay_lb`

Wait `delay` times 10 ms until the currently executing macro will be continued. The maximum value for `speed` is 65535 (Hex FFFF) will give a maximum delay of close to 11 minutes. The only way to terminate such a delay is to send any command (such as the "No Operation" command – see page 16) to the iLCD controller. This is the reason why delays are ignored in normal command mode and why entering the command does not have any effect when not called via a macro.

Set Macro Execution Speed

`<CID> O S speed_hb speed_hb`

This command can be used to simulate a typewriter. This means that the bytes of the command which follows the Set Macro Executions Speed command will each be executed at intervals of speed times 10ms. The maximum value for `speed` is 65535 (Hex FFFF) and this will give a maximum delay of close to 11 minutes for processing one command character.

Macro execution speed settings are ignored in normal command mode and entering the command does not have any effect when not called via a macro.

Macro execution speed is automatically reset when all macros are finished.

Allow Keyboard/Touch Macros To Start

`<CID> O K`

Touch screen macros (see "The Concept of iLCD's Touch Fields" on page 42) called automatically when a touch field is pressed/released usually will not be stopped by any other touch screen macro to avoid partially drawn touch fields when a new touch field is pressed or the previous one is released before the macro has been finished. The iLCD controller carries out this behavior automatically. If you want to enable the iLCD controller to react to a new touch screen macro before the actual one is completed, insert the above sequence into your macro.

A typical case for using this command is as follows. At the end of the drawing stuff carried out in the break macro of a touch field a delay might need to be started before jumping to a "timeout" macro. Normally no other touch macros would be executed until this delay is finished (causing a "dead" touch screen), but when executing the above command before the delay, any newly pressed touch field aborts the current macro (causing the delay to stop) and executes the macro attached to the new touch field.

Please note that this command is ignored when not called via a macro.

Set Macro Timer

`<CID> O T time`

Define a time period `time`, in which incoming commands are buffered so that running macros will not be aborted. This enables calling macros with subsequent command lines from the iLCD Manager's Terminal and can be used to prevent macros from being stopped by any `<CID>` character. When macro execution is finished, the subsequent commands will be executed immediately regardless of the remaining time.

Note, that the timer is only started, if the macro is executed by an incoming command. When a macro is called by another macro, the timer is not restarted, but keeps running if there is time left.

`time` is a byte value between 0 and 50, the resolution is 100ms. So for example, a `time` of 10 represents a period of 1 second.

Touch Screen Related Commands

The Concept of iLCD's Touch Fields

To allow maximum flexibility and easy use of touch fields without using any resources of the application's controller, the following concept was implemented:

You may define up to 64 touch fields per screen with any rectangle size and where fields may overlap each other. The maximum field size is the pixel screen size used by the actual iLCD. Any touch field may have a key assigned which is reported to the application when the touch field is pressed and released. In addition, any touch field may have a text template assigned to it (the text to be drawn within the field, defined via the setup software) and a make and/or a break macro which are automatically executed when the field is pressed/released.

The iLCD controller automatically checks in the order of field indices if the touch panel is pressed within one of the previously defined fields. If the touch pressure is within the bounds of a touch field, the make key sequence is reported to the application if a key is assigned to this field. This key sequence is exactly the same sequence as used for a keyboard – see at "Touch Field Press/Release" on page 65. After the key has been reported, the make macro assigned to the field is executed. Before execution of this macro (defined via the setup program) starts, the iLCD controller sets the touch field index accordingly and the screen coordinate to the upper/leftmost corner of the touch field, allowing the execution of all necessary drawing stuff within the make macro (usually painting the touch field in a "pressed" state). After the make macro has been fully executed (or an "Allow Keyboard/Touch Macros To Start" command - see page 41 - has been found), the iLCD controller checks if the touch screen has been released or waits for release. On release the corresponding break key sequence is sent to the application and the break macro is executed.

The iLCD controller also keeps track of which touch field has been pressed using an internal variable known as the 'current touch field index'. This is automatically set when a touch field is pressed but the user can also set it manually by using Set Touch Field Index. This facility is useful if you want to call a common macro from a number of touch fields. In this case you won't know in advance the index of the touched field but by calling a command such as Execute Touch Make Macro with the parameter hex FF (decimal byte –1) the command will call the macro appropriate to the touched field. Another example is the use of a common macro to define a number of touch fields. In this case, for each touch field in turn, you can perform any setup unique to that field (such as screen location), define the current touch field index using "Set Touch Field Index", then finally call the setup macro to create the touch field.

Please note that the "basic" touch field does not carry out any screen drawing, it only sends keys sequences and calls the make/break macro thus allowing maximum flexibility. One Make/break macro can be used for any number of different touch fields on different locations, as the screen coordinate is set automatically before the macro starts, and painting the touch field text can be done via a special command which uses the touch field index set be the iLCD controller in advance as well. Cursor movements within make/break macros can be done relative to the actual screen coordinate via the "Increment/Decrement Column Address" (see page 20) and "Increment/Decrement Row Address" (see page 20) to allow the use of the macro for different touch fields on different screen coordinates. Using the "Cursor Memory Related Commands" (see page 40) can further help you to simplify the field drawing routines and to increase the usability of macros by multiple touch fields.

Please keep in mind, that any make/break macro has to be fully executed before the touch panel processing can be continued except for the special case when using the "Allow Keyboard/Touch Macros To Start". This is why drawing the shape of the touch field should preferably be done via painting an image (or setting the frame number of an animated image) rather than, for example, drawing a rectangle with shadow since outputting images is much faster than executing graphic routines. When drawing the text of a touch field via the special "Draw Touch Field Text" (see page 46) command you may embed any sequence in the text template, giving you the maximum freedom of how to render your touch field text labels: You may even use multi-line text with different fonts with one macro without having to take care which field is to be drawn.

1 Only firmware versions ≥ 2.0 support touch fields and older versions will report an error when trying to use one of the touch field commands. You may update your iLCD Controller's firmware version via the iLCD setup software accordingly if necessary.

Calibrate Touch Screen

```
<CID> T C
```

Before using any touch screen related commands a calibration of the touch panel has to be carried out. All iLCD panels with touch panel are already calibrated at production time but you may repeat the calibration process anytime. When issuing this command, the iLCD panel requests that you press the upper leftmost and lower rightmost pixel position. After pressing these two positions, the calibration is done and the calibration values are stored in the iLCD controller automatically. Sending the "Reset All" command (see page 16) stops the calibration process, the previous existing values are used then.

Please note that reverse connections of the X and Y touch panel (swapped left/right and/or top/bottom contacts) are detected automatically and corrected accordingly. The iLCD setup software allows you to swap the horizontal/vertical touch panel direction by enabling the appropriate check box.

Calibrate Touch Screen and Report

```
<CID> T c
```

This command starts the touch screen calibration just as "Calibrate Touch Screen". When the calibration is done, the iLCD panel sends a report (see "Calibrate Touch Screen Done" on page 64 for the format).

Verify Touch Screen Calibration

```
<CID> T V
```

The calibration of the touch screen can be tested via this command. Every touch position is marked with a small cross on the screen, the validation can be stopped via the "Reset All" command (see page 16).

Set Touch Field Width

`<CID> T W width_hb width_lb`

This command sets up the width of any subsequently defined touch field (see the "Create/Define Touch Field" command on page 45) to `width`. Please note that sending this command does not change anything for touch fields already defined. After power up and after sending the "Reset All" command (see page 16) `width` is set to 30.

Set Touch Field Height

`<CID> T H height_hb height_lb`

This command sets up the height of any subsequently defined touch field (see the "Create/Define Touch Field" command on page 45) to `height`. Please note that sending this command does not change anything for touch fields already defined. After power up and after sending the "Reset All" command (see page 16) `height` is set to 20.

Set Touch Field Make Macro

`<CID> T M macro_idx_hb macro_idx_lb`

This command assigns the macro to be called when a touch field is pressed of any subsequently defined touch field (see the "Create/Define Touch Field" command on page 45) to `height`. Please note that sending this command does not change anything for touch fields already defined. Setting `macro_idx` to hex FFFF (decimal word –1) causes no macro to be executed when the subsequently defined touch field is pressed. After power up and after sending the "Reset All" command (see page 16) `macro_idx` is set to -1. Using a `macro_idx` that has not previously defined by the setup software causes a <NACK> to be sent by the iLCD controller.

Set Touch Field Break Macro

`<CID> T B macro_idx_hb macro_idx_lb`

This command assigns the macro to be called when a touch field is pressed of any subsequently defined touch field (see the "Create/Define Touch Field" command on page 45) to `height`. Please note that sending this command does not change anything for touch fields already defined. Setting `macro_idx` to hex FFFF (decimal word –1) causes no macro to be executed when the subsequently defined touch field is pressed. After power up and after sending the "Reset All" command (see page 16) `macro_idx` is set to -1. Using a `macro_idx` that has not previously defined by the setup software causes a <NACK> to be sent by the iLCD controller.

Set Touch Field Text Template Index

`<CID> T T text_template_idx_hb text_template_idx_lb`

This command assigns the text template index of any subsequently defined touch field (see the "Create/Define Touch Field" command on page 45) to `template_idx`. Please note that sending this command does not change anything for touch fields already defined. Setting `template_idx` to hex FFFF (decimal word –1) causes no text template to be assigned to the subsequently defined touch field. After power up and after sending the "Reset All" command (see page 16) `template_idx` is set to -1. Using a `template_idx` that has not previously defined by the setup software causes a <NACK> to be sent by the iLCD controller.

Text templates assigned to a touch field are used by the special "Draw Touch Field Text" (see page 46) based on the current touch field index (see "Set Touch Field Index" on page 45). The touch field index is set automatically before a make or break macro is called. Please note that the text template can even contain any valid ANSI sequence so allowing various attributes/fonts to be changed easily.

Create/Define Touch Field

```
<CID> T A field_idx key
```

This command creates or redefines the touch field with index `field_idx` (0…63) with the current values previously set via the "Set Touch Field Width", "Set Touch Field Height", "Set Touch Field Make Macro", "Set Touch Field Break Macro" and "Set Touch Field Text Template Index" commands and assigns `key` to the field. The upper left position of the touch field is taken from the current cursor position. Assigning a `key` value of 0 will inhibit the reporting of the make/break sequence to the application. The make/break macros, if assigned, will still however be executed.

`field_idx` may have the hex value FF (decimal byte –1) and the controller will choose the next free (unassigned) touch field index.

Using an invalid `field_idx` or a value of FF when all 64 touch fields are already defined causes a <NACK> to be sent by the iLCD controller.

Please note that this command can even be used within a make or break macro to redefine the behavior of the current touch field to implement, for example, a toggle field behavior.

Using this command also sets the current touch field index, which is used by "Remove Touch Field" (see page 45), "Set Touch Field Index" (see page 45), "Execute Touch Make Macro" (page 46), Execute Touch Break Macro (page 46) and "Draw Touch Field Text" (see page 46), if the value `field_idx` for these commands is set to FF (decimal byte –1).

Remove Touch Field

```
<CID> T R field_idx
```

Remove the touch field with index `field_idx`. If `field_idx` is lower 0 or greater 63, the iLCD controller sends a <NACK>. If `field_idx` is set to hex FF (decimal byte –1), ALL touch fields are removed.

Please note that executing the "Reset All" command (see page 16) removes all touch fields as well.

Global En/Disable Touch Fields

```
<CID> T G on_off
```

Enables (`on_off` = 1) or disables (`on_off` = 0) all touch fields. The definition of any touch field is not changed by this command, the fields are simply ignored when they are disabled. Disabling touch fields until all touch fields are defined can be a sensible precaution because it prevents the possible interruption of any macro used to define several fields

Please note that executing the "Reset All" command (see page 16) removes all touch fields and globally enables touch field processing then.

Set Touch Field Index

```
<CID> T I field_idx
```

Set the iLCD Controller's current touch field index to `field_idx`. If `field_idx` is lower than 0 or greater than 63, the iLCD controller sends a <NACK>.

Please note that the iLCD Controller's current touch field index is set automatically when a touch field is pressed or released.

The value of the current touch field index is used by "Remove Touch Field" (see page 45), "Set Touch Field Index" (see page 45), "Execute Touch Make Macro" (page 46), Execute Touch Break Macro (page 46) and "Draw Touch Field Text" (see page 46), if the value `field_idx` for these commands is set to FF (decimal byte –1).

Execute Touch Make Macro

```
<CID> T E M field_idx
```

Execute the make macro assigned to touch field with index `field_idx`. If `field_idx` is set to hex FF (decimal byte –1) the current touch field index is used. If `field_idx` is out of bounds, the index has no field attached yet or the touch field indexed has no make macro assigned to, a <NACK> is sent by the iLCD controller.

Please note that calling a make macro is normally done automatically via the iLCD controller when the appropriate touch field is touched, but can be done via the above command as well.

Execute Touch Break Macro

```
<CID> T E B field_idx
```

Execute the break macro assigned to touch field with index `field_idx`. If `field_idx` is set to hex FF (decimal byte –1) the current touch field index is used. If `field_idx` is out of bounds, the index has no field attached yet or the touch field indexed has no break macro assigned to, a <NACK> is sent by the iLCD controller.

Please note that calling a make break is normally done automatically via the iLCD controller when the appropriate touch field is touched, but can be done via the above command as well. This can be useful when, for example, drawing the "not down" state of a touch field after having defined the field.

Draw Touch Field Text

```
<CID> T D field_idx
```

Draw the text template assigned to touch field `field_idx`. If `field_idx` is set to FF (decimal byte –1) the current touch field index is used. This index is automatically set when a touch field is pressed or released, but can be set via the "Set Touch Field Index" command (see page 45) as well.

If no text template is assigned to the field and the key assigned to the field is other than Hex 0, the key is printed instead. This behavior is useful for single character touch fields like "keyboard" fields, as no text template has to be assigned for every touch field.

1 This "shortcut"-behavior is available for firmware versions ≥ 2.09 only.

If `field_idx` is out of bounds, the corresponding index has no active field attached or no text can be displayed, a <NACK> is sent by the iLCD controller.

Please note that when using this command within a make or break macro, the screen coordinate is automatically set to the upper/left point of the corresponding touch field.

Normally the text contained in the text template assigned to the field is outputted. If there is no text template assigned, the key code, if any, of the field is outputted instead.

Enable/Disable Reporting Touch-Coordinates

<CID> T K onoff

When reporting a pressed or released touch field normally only the key is reported as described in "Touch Field Press/Release" on page 65. Under certain circumstances the coordinates of pressing and releasing the field is of interest as well. When sending the above command with `onoff` set to Hex 01, all future pressure/release events of touch fields having a key assigned send the coordinates of the event as well as shown in "Touch Field Press/Release + Coordinate of Event" on page 65. When `onoff` is Hex 00, the normal key report is turned on again. Please note, that sending the "Reset All" (page 16) command enables normal touch field reporting as well.

Please note that this command does not work with firmware versions lower than version 1.12.

Enable/Disable Reporting Movements

<CID> T O onoff

When creating touch fields where the actual location of the pressure is of interest even when this location is moving (without releasing the touch field), such as when scrolling a scrollbar, this movement can be reported using the above command with `onoff` set to Hex 01. The report of the movement is shown in "Moving Coordinate of Touch-Field Press" on page 65. Please note that only touch fields having a key attached get reported. When `onoff` is Hex 00, the movements reporting is turned off. Please note, that sending the "Reset All" (page 16) command disables movements reporting as well.

Please note, that this command does not work with firmware versions lower than version 1.12.

Retrieve Last Touch Screen Event

<CID> T ?

When a previously defined touch field (see at "Create/Define Touch Field" on page 45) is pressed or released or the position of pressure is changed, this event is saved in the iLCD controller even when no key is assigned to the touch field. So even when the touch field event and/or coordinate information is not reported, the most recent event can be retrieved at anytime. The data returned is described in chapter "Touch-Field Event" on page 64.

Please note, that this command does not work with firmware versions lower than version 1.12.

Input/Output Related Commands

General Information About Inputs/Outputs

The DPC20xx and DPC3020 controllers allows most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via the setup software. All commands referring to port pins below refer to the logical port name, not the physical port pin name.

This means the physical port pin "Keyboard column 3" may have the logical name e.g. "Output #9", so turning on this pin (= setting it to high when it is defined as a push/pull pin) can be done via the "Set LED" command (see page 48)

<CID> I L S 09_H 01_H

When using the same port as a pull-down output port, sending the above command pulls the pin to low (making it "active") instead. This is the behavior the DPC10xx controller has by default. Note that the DPC20xx and DPC3020 controllers can only source/sink 4mA. Please check the documentation for your particular board.

In the commands relating to LEDs the port pins LED 0, LED 1 … are identical to the corresponding pins "Output #0", "Output #1"… The terms LED 0, LED 1 … have been kept the same to maintain upwards compatibility.

Set LED

```
<CID> I L S led_no mode
```

where `led_no` can have a value between 0 and 5 (for [1]) respectively 15 (for [2][3]) (describes the LED / output number to be turned on/off - see also "Set Multiple LEDs" on page 48) and `mode` is as follows:

`mode` = 0	sets LED to off
`mode` = 1	sets LED to on
`mode` = 2	sets LED to blinking

The startup values for all LEDs / outputs are defined via the setup program.

[1] When trying to switch on/off the LEDs that are connected to a general purpose I/O pin, which is not defined as a LED output via the setup program, the command is simply ignored although an <ACK> is returned.

[2][3] When trying to address an output port that is not defined as an output via the setup program, the command is simply ignored; an <ACK> is returned then.

Set Multiple LEDs

[1] DPC10xx only

```
<CID> I L s led_mask blink_mask
```

where `led_mask` and `blink_mask` consists of the following bits:

Bit 0	LED 0
Bit 1	LED 1
Bit 2	LED 2
Bit 3	LED 3
Bit 4	LED 4 (Power LED)
Bit 5	LED 5

This gives a maximum value of Hex 3F for `led_mask` and `blink_mask`. If the corresponding bit in `led_mask` is one, the LED is on, if the corresponding bit in `blink_mask` and `led_mask` is one, the LED blinks.

Please note that LED 4 and LED 5 share the same controller outputs as column 2 and 3 of the keyboard. Although there is no visual effect when using LEDs, there are short glitches on the corresponding outputs when the keyboard is scanned, so it is not recommended that these outputs be used for something other than LEDs.

LED 4 is turned on automatically when Smart Power (see "Set Smart Power-Off Mode" on page 59) is set to on, and the power key is pressed without issuing any command to the iLCD controller.

The startup values for all LEDs are defined via the setup program. Bits referring to the LEDs connected to a general purpose I/O pin which is not defined as a LED output via the setup program are ignored by the iLCD controller.

2 3 DPC20xx and DPC3020

```
<CID> I L s led_mask_hb led_mask_lb blink_mask_hb blink_mask_lb
```

`led_mask` and `blink_mask` refer to the bit addressed outputs. Bit 0 refers to output #0, bit 15 to output #15. If a bit set to one is not previously defined as an output via the setup software, setting this bit has no effect.

The startup values for all output ports are defined via the setup program.

Set LED Blink Frequency

```
<CID> I L F frequency
```

`frequency` is a value between 1 and Hex FF and describes the number of 10 ms intervals until the state changes from on to off and *vice versa*. This means that a value of decimal 25 (Hex 19), for example, gives a frequency of 2 Hertz (250ms on and 250ms off). The default blink frequency is set via the setup program. When a LED is currently in blink mode, the frequency changes accordingly.

Set Relays On/Off/PWM

```
<CID> I R relay_no mode
```

Switch on (`mode` = 1) or off (`mode` = 0) relay number `relay_no` (0…1). Setting mode to 2 enables the PWM #0 (for relay output 0) or PWM #1 (for relay output 1). The corresponding relay output pulses with the PWM signal according to "Pulse Width Modulation (PWM) Related Commands" on page 53 instead of being statically switched to on or off.

1 DPC10xx only

When using a hardware release lower than 3.0, mode can only range from 0-1. The PWM functionality is not provided on these iLCD controllers.

Relays One Shot

```
<CID> I r relay_no mode time_hb time_lb
```

Switch on (`mode` = 1) or off (`mode` = 0) relay number `relay_no` (0…1) for `time` times 10 ms. The maximum value for `time` is 65535 (Hex FFFF) will give a maximum one-shot duration of close to 11 minutes. When using a iLCD controller with hardware release 3.0 or higher `mode` may also set to 2, which enables the PWM #0 (for relay output 0) or PWM #1 (for relay output 1) for `time` times 10 ms. The corresponding relay output pulses with the PWM signal according to "Pulse Width Modulation (PWM) Related Commands" on page 53 instead of being statically switched to on or off for the selected time.

Enable Keyboard

```
<CID> I K E on_off
```

If you do not want the keyboard to be scanned (for example, to prevent the glitches on LED 4 and 5 output when no keyboard is connected) keyboard scanning can be turned off by setting `on_off` to 0. Re-enabling the keyboard can be done by setting `on_off` to 1. Please note that enabling/disabling keyboard does not change keyboard reporting (see "Enable Keyboard Report" on page 50), although a disabled keyboard will not report any keys.

The startup value of keyboard enabling is set via the setup program.

Enable Keyboard Report

`<CID> I K R on_off`

When keyboard reporting is on (`on_off` = 1), any key pressure and release gets reported (see the data response at "Key Press/Release" on page 65) automatically and any running macro is stopped. When the terminal mode (see "Terminal Mode" on page 3) is on, only the key itself gets reported when the key is pressed (this is what we expect from a standard ANSI terminal).

Switching off keyboard reporting (on_off = 0) avoids stopping macros. The startup value of keyboard reporting is set via the setup program.

Get Keyboard State

`<CID> I K ?`

The state of all keys can be retrieved asynchronously by using this command. See at "Keyboard State" on page 63 to learn about the structure of the reported data.

Set Baud Rate

Under some certain circumstances, it may be useful to change the communication baud rate "on the fly", although the baud rate is usually set via the setup program. Please note that restarting the iLCD controller via the "Reboot Panel Controller" command (page 17) or via the reset pin causes the controller to start with the baud rate set via the setup program.

1 DPC10xx only

`<CID> I B divisor_hb divisor_lb`

The values for divisor are calculated as follows (C-syntax, all values in long):

```
divisor = (word) (0x10000L - (((OSC_FREQ * 125L * 10L) / (4L * (baud_rate)) + 5L) / 10L))
```

where `OSC_FREQ` is `36000L` (the crystal frequency used on the standard iLCD controller – when using the 3V types a different oscillator frequency will be used, please contact support@demmel.com when you use a 3V controller type) and `baud_rate` is the baud rate to be set.

Some standard baud rates are as follows:

Baud Rate	Divisor (dec.)	Divisor (Hex)
115200	65526	FFF6
57600	65516	FFEC
38400	65507	FFE3
28800	65497	FFD9
19200	65477	FFC5
14400	65458	FFB2
9600	65419	FF8B
7200	65380	FF64
4800	65302	FF16
2400	65067	FE2B
1200	64598	FC56
600	63661	F8AD
300	61786	F15A
150	58036	E2B4

DPC20xx and DPC3020

```
<CID> I B port_no baud_b3 baud_b2 baud_b1 baud_b0
```

The baud rate `baud` is sent as a long value consisting of 4 bytes (see "32-Bit Values" on page 4. The user does not have to take care about divisors and oscillator frequencies when using the DPC20xx and DPC3020 iLCD controllers.

`port_no` describes the serial port for which the baud rate is to be changed. When using the value 0, the serial port currently in use is changed, a value of 1 refers to serial port #0 and a value of 2 refers to serial port #1. If `port_no` is 0, and the communication port in use is not a serial port, the command is ignored.

Note for color iLCD panels

If your color iLCD panel with a DPC3020 controller has a USB port installed, it has a USB to serial bridge on board where the USB port is connected to the serial port 0 of the iLCD controller. Changing the baud rate of serial port 0 changes the transmission speed between the USB bridge and the iLCD controller, the serial baud rate of the virtual COM port used for communicating with the iLCD panel has to be changed accordingly.

Get Current Communication-Port

```
<CID> I ? C
```

Retrieves the communication port in use. The answer to this command is as follows:

```
<ACK> port_id <ACK>
```

where `port_id` is as follows:

01_H	Serial port 0
02_H	Serial port 1
03_H	USB port
04_H	I^2C port
05_H	SPI port

Note for color iLCD panels

If your color iLCD panel has an USB port installed, the above command will return 00_H indicating serial port 0 as this board has a USB to serial bridge on board, where the USB port is connected to the serial port 0 of the iLCD controller. Communication via this USB port is done out via the virtual COM port of the USB bridge, not via "real" USB communication.

Get Inputs State

```
<CID> I ? I
```

DPC10xx only

The 4 general I/O ports can be defined as LED outputs, digital inputs or ADC inputs via the setup program. The current state of any of the 4 general I/O ports defined as a digital input can be retrieved via this command. The iLCD controller returns `<ACK> state <ACK>` where the bits of state reflect the high/low state of the corresponding input (bit 0 = general I/O port 0 … bit 3 = general I/O port 3).

General I/O ports not defined as a digital input are report as low, while digital inputs, which are left unconnected, are reported as high.

2 3 DPC20xx and DPC3020

The iLCD controller returns <ACK> state_hb state_lb <ACK> where the bits of state reflect the high/low state of the corresponding input port pins defined as a digital input via the setup software. The inputs are reported in a bit orientated manner, where bit 0 of state refers to Input #0 and bit 15 refers to Input #15. Undefined inputs are reported as low, pins defined as inputs but left unconnected (=floating) can have any value.

Get ADC Value

```
<CID> I ? A port
```

1 DPC10xx only

The 4 general I/O ports can be defined as LED outputs, digital inputs or ADC inputs via the setup program. The current value of any of the general 4 I/O ports defined as ADC (Analog Digital Converter) input can be retrieved via this command.

The iLCD's ADCs have a resolution of 8 bits, resulting in a value range of 0...255 where 0 is an input voltage of 0V and 255 is an input voltage of 5V. The actual upper value read is dependent upon the supply voltage of the iLCD module which can vary between 4.9V and 5.1V.

All iLCD modules can be supplied with an alternative input voltage range of 0V … 2.5V or 0V … 4V by adding a correspondent micro power reference. If you want to use this option please contact demmel products via support@demmel.com.

The iLCD controller returns <ACK> value_hb value_lb <ACK> where value has a range of 0 … 255 (sending a 16 bit value is implemented due to possible future extensions of the iLCD's ADC resolution).

If channel has a value > 3 or the corresponding port is not defined as an ADC input, a <NACK> is returned.

2 3 DPC20xx and DPC3020

The 4 general I/O ports GP0 to GP3 can alternatively be defined ADC inputs via the setup program. The current value of any of the general 4 I/O ports defined as ADC (Analog Digital Converter) input can be retrieved via this command.

The iLCD's ADCs have a resolution of 10 bits, resulting in a value range of 0...1023_D ($1FF_H$) where 0 is an input voltage of 0V and 1023_D is an input voltage of 3.3V, the value of the supply voltage of the iLCD controller.

Time/Date Related Commands

Please note, that date/time related commands can only be used on iLCD panels installed with a real time clock (RTC) option. In any other case the values read back by "Get Time" and "Get Date" can have indeterminate values. When the RTC is backed up via a battery (either on-board or external) during power loss, the RTC counts the time/date information even without any supply voltage applied.

Please note, that time/date related commands do not work on firmware versions < V1.19.

Set Time 2 3

```
<CID> I T hour minute second
```

Sets the time of the real time clock.

The following byte-variables apply:

```
hour      0 ... 23
minute    0 ... 59
second    0 ... 59
```

Get Time

```
<CID> I ? T
```

When sending this command, the current time is returned in the following format:

```
<ACK> hour minute second <ACK>
```

Please have a look at "Set Time" on page 52 to learn about the range of the above mentioned byte-values.

Set Date

```
<CID> I D year month day weekday
```

Sets the date of the real time clock.

The following byte-variables apply:

```
year      0 ... 99 (indicates year 2000 ... 2099)
month     1 ... 12
day       1 ... 31
weekday   0 ... 6
```

Please note, that there is no connection between the actual date and the value weekday. The weekday value is simply incremented every day, so it's up to the user to assign a weekday-number to a day such as 0 = Sunday.

Leap years up to 2099 are taken into consideration when the date is incremented by the RTC automatically.

Get Date

```
<CID> I ? D
```

When sending this command, the current date is returned in the following format:

```
<ACK> year month day weekday <ACK>
```

Please have a look at "Set Date" on page 53 to learn about the range of the above mentioned byte-values.

Pulse Width Modulation (PWM) Related Commands

The iLCD controllers with hardware release 3.0 and higher (for DPC10xx, all hardware releases for DPC20xx and DPC3020) can control the two relay outputs via two internal separate pulse-width modulation circuits too. This allows driving a speaker or buzzer via relay output 0 to produce a sound like mobile phone's ring tones (use macros to control the sequences of different tone heights and durations). Using a simple R/C filter on any of the two relay outputs can allow the output of a controllable analog voltage as well.

1 DPC10xx only

Relay output 1 can use the fixed frequency (approximately 7.81 kHz) PWM #1 with variable Duty Cycle and negative Polarity only, relay output 0 can be controlled via PWM #0 with much more functionality:

PWM #0 can be programmed to provide a PWM output with variable pulse width and period. It has a 16-bit Prescaler, an 8-bit Counter, a Pulse Width Register, and a Period Register. The Pulse Width Register defines the PWM pulse width time, while the Period Register defines the period of the PWM. The input clock to the Prescaler is the controller's XTAL frequency (f_{OSC} = 36.000 MHz for all 5V iLCD controllers) divided by 2. The 16-bit Prescaler1 divides the input clock ($f_{OSC}/2$) to the desired frequency, the resulting clock runs the 8-bit Counter of the PWM #0.

The input clock frequency to the PWM #0 Counter is:

$$f\ PWM\ \#0 = (f_{OSC}/2)/(\text{Prescaler data value} + 1)$$

When the Prescaler is set to data value '0,' the maximum input clock frequency to the PWM # 0 Counter is fOSC/2. The PWM #0 Counter is a free-running, 8-bit counter. The output of the counter is compared to the Compare Registers, which are loaded with data from the Pulse Width Register and the Period Register. The Pulse Width Register defines the pulse duration or the Pulse Width, while the Period Register defines the period of the PWM # 0. When the PWM is enabled, the register values are loaded into the Comparator Registers and are compared to the Counter output. When the content of the counter is equal to or greater than the value in the Pulse Width Register, it sets the PWM #0 output to low (with Polarity = 0). When the Period Register equals to the PWM #0 Counter, the Counter is cleared, and the PWM #0 output is set to logic 'high' level (beginning of the next PWM pulse). The Period Register cannot have a value of "00" and its content must be greater than the Pulse Width Register. The Prescaler Register, Pulse Width Register, and Period Register can be modified while the PWM channel is active. The values of these registers are automatically loaded into the Prescaler Counter and Comparator Registers when the current PWM #0 period ends. The same is true for PWM #1 when changing the Duty Cycle on the fly. Setting Polarity (only available for PWM #0) to 1 inverts the output of relay 0.

Please note that the iLCD PWM controller outputs are inverted on standard iLCD modules via the relay driving open collector transistor stage.

2 3 DPC20xx and DPC3020

Relay output 1 can use the fixed frequency (1.0 kHz) PWM #1 with variable duty cycle only, relay output 0 can be controlled via PWM #0. The frequency of PWM #0 can be set between 1 Hz and 1 MHz although 100 kHz should not be exceeded when using an external transistor stage, as done in most of the iLCD boards.

Set PWM #0

1 DPC10xx only

`<CID> I P 00`$_H$ `prescaler_hb prescaler_lb pulse_width period polarity`

`prescaler_hb` * 256 + `prescaler_lb` sets the prescaler value, `pulse_width` sets the Pulse Width Register, `period` sets the Period Register and `polarity` sets the Polarity according to "Pulse Width Modulation (PWM) Related Commands" on page 53 of PWM #0.

At power up the variable values are set as follows:

`prescaler`	decimal 70
`pulse_width`	decimal 127
`period`	decimal 255
`polarity`	1

Giving a PWM input clock frequency of 257.14 kHz and an output frequency of 1kHz with a duty cycle of 50% and negative polarity.

DPC20xx and DPC3020

`<CID> I P` 00_H `freq_b3 freq_b2 freq_b1 freq_b0 duty_cycle_hb duty_cycle_lb`

where `freq` can have a value between 1 and 1,000,000 (1 Hz to 1 MHz) – see "32-Bit Values" on page 4 – and `duty_cycle` must be in the range of 1 to 9999_D. A `duty_cycle` of 1 cause the relay output to be switch on for 0.1‰ only, a value of 9999_D the output is switched on for 99.99% of the period and a value of 5000_D sets a duty cycle of 50%.

Please note that the extra output transistor stage contained on most iLCD boards inverts the signal, so the duty cycle is inverted as well in this case.

On power up `duty_cycle` is set to 50% duty cycle automatically.

Set PWM #1

DPC10xx only

`<CID> I P` 01_H `duty_cycle`

`duty_cycle` sets Duty Cycle according to "Pulse Width Modulation (PWM) Related Commands" on page 53 of PWM #1.

On power up `duty_cycle` is set to Hex 80 (equals 50% duty cycle).

DPC20xx and DPC3020

`<CID> I P` 01_H `duty_cycle_hb duty_cycle_lb`

`duty_cycle` of the fixed-frequency PWM # 1 with 1kHz frequency must be in the range of 1 to 9999_D. A `duty_cycle` of 1 cause the relay output to be switch on for 0.1‰ only, a value of 9999_D the output is switched on for 99.99% of the period and a value of 5000_D sets a duty cycle of 50%.

Please note that the extra output transistor stage contained on most iLCD boards inverts the signal, so the duty cycle is inverted as well in this case.

On power up `duty_cycle` is set to 50% duty cycle automatically.

EEPROM Related Commands

The iLCD controllers contain an EEPROM emulation behaving like a real EEPROM, which means that you can write any value into any memory location at any time. This is different to usual Flash memory behavior, which must be erased before overwriting a value. The only difference to a real EEPROM is, that when issuing the "format" command (erasing all locations to hex FF), and after some number of updates, the write process may take up to one or two seconds. During this period no other commands can be carried out. For example, blinking LEDs or animated graphics are stopped. Any data sent to the iLCD controller during this peeriod will be lost as well.

The EEPROM emulation provides 512 (DPC10xx) / 240 (DPCX20xx) / 496 (DPCX3020) byte space; only the first 3 (4 on DPC3020) values are used by the iLCD controller itself, however these values can be freely written to. Any other location is not used by the iLCD controller and may be used for the controlling application to store and retrieve its private data which is kept, even after a power loss.

The 3 special values are defined as follows:

Location	Contents	Range Of Values	Reference Description	Page
0	LCD Contrast	0 … 255	Set LCD Contrast	25
1	Backlight Intensity	0 … 15	Set Backlight Intensity	25
2	Backlight Mode	0 … 2	Get Backlight Mode	24
3	LCD Gamma Value	0…255	Set LCD Gamma Value 3	26

Although any value can be written to any location, the iLCD controller will automatically restrict values to the corresponding range when retrieving data from the EEPROM.

The 3 (4) values mentioned above are restored from the EEPROM and the corresponding internal commands (e.g. for setting the backlight intensity) are carried out at startup (also when a "Reboot Panel Controller" – see at page 17 – is issued) and when a "Reset All" (page 16) or a "Reset All and Show Startup Graphic" command (page 17) is sent to the controller.

Please note that writing to the 3 (4) special EEPROM locations via the "Write EEPROM" command (page 56) does not call the associated command. The new settings will only become active upon start-up. When you want to set something like the backlight intensity, use the appropriate command instead of – or in addition to – the "Write EEPROM" command.

Get EEPROM Size 2 3

```
<CID> E ?
```

Retrieves the size of the EEPROM.

The command returns

```
<ACK> eeprom_size_hb eeprom_size_lb <ACK>
```

Erase EEPROM

```
<CID> E E =
```

When the EEPROM is erased (all values are written to hex FF) the 3 (4) special locations (see "EEPROM Related Commands" on page 55) are loaded from the Flash data after the EEPROM has been formatted. The appropriate values in the Flash memory are set via the setup program.

Read EEPROM

```
<CID> E R index_hb index_lb
```

Read the contents of EEPROM location `index,` where `index` has a value range of 0 to the size of the EEPROM – 1. The value is returned as `<ACK> value <ACK>`. Any location not previously written to will return hex FF.

Write EEPROM

```
<CID> E W index_hb index_lb value
```

Write `value` to the EEPROM location `index`, where `index` has a value range of 0 to the size of the EEPROM – 1. If there is a write error, an <EERR> (Hex 10) response instead of <ACK> is sent back. See some further explanations about the case on "EEPROM Write Error" (page 60).

Copyright © by demmel products, Document Date: December 2, 2009

Power/Watchdog Related Commands

General Information About Power/Watchdog Related Inputs/Outputs

The DPC20xx and DPC3020 controllers allow most port pins to be assigned as digital or analog inputs, outputs (pull down or push/pull) or keyboard columns via the setup software. All commands referring to port pins below refer to the logical port name, not the physical port pin name.

The logical port names dealing with Power/Watch Dog Related Inputs/Outputs are as follows:

ARES	Watch dog reset output
APWR	PC power off output
ASPWR	Disconnect PC's power switch output
APSWI	Input from the PC's power switch

If a port function is not previously defined via the setup software, the corresponding command is inactive. So, for example, when running the watchdog the ARES function must be assigned to a physical port pin via the setup software to enable the pin to go high when the watchdog triggers.

Set Watchdog Interval

```
<CID> P W interval_hb interval_lb
```

Sets the watchdog interval to `interval` times 10 ms thus allowing a maximum of 655.35 seconds (10.92 minutes). The current watchdog interval is retriggered. A value of 0 disables the watchdog. Please note that quoting must be done if either `interval_hb` or `interval_lb` contains a value, which is equal to `<CID>`.

DPC10xx only

If the PC fails to trigger the watchdog before its time is expired, the panel controller pulls the main board's reset pin low for a certain time (defined via the setup program), disables the watchdog, sets the LEDs to the power up state, shows "`Watchdog Reset`" (may be overwritten via the setup program) on the LCD display and sets the backlight to power up value.

At startup the watchdog is disabled.

DPC20xx and DPC3020

If the PC fails to trigger the watchdog before its time is expired, the panel controller activates the port pin assigned to the ARES function for a certain time (defined via the setup program), disables the watchdog and shows the watchdog reset message box (defaults to "`Watchdog Reset`" and set via the setup program) on the LCD display. If the watchdog-reset message is empty, no watchdog reset message box is displayed. If no pin is assigned to the ARES port function, no hardware activity will be carried out on watchdog reset.

At startup the watchdog is disabled.

Trigger Watchdog

```
<CID> P w
```

If the watchdog is enabled, triggering allows the application to further process another watchdog interval without pulling the reset line. If the watchdog is disabled this command does nothing although it is acknowledged by an <ACK> command.

Shutdown (Power Off)

<CID> P U =

DPC10xx only

Pulls the main board's power down for a short time (set via the setup program). The display shows "Shutting Down.." (may be overwritten via the setup program). The '=' (Hex 3D) character is used to avoid accidental triggering of the shutdown sequence.

DPC20xx and DPC3020

Activates the port pin with function APWR assigned to for a short time (set via the setup program). The display shows the shutdown message box (defaults to "Shutting Down.." and set via the setup program). If the shutdown message is empty, no shutdown message box is displayed. If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.

Hard Shutdown (Long Power Off)

<CID> P u =

DPC10xx only

Pulls the main board's power down for a long period (set via the setup program). The display shows "Hard Shutdown" (may be overwritten via the setup program). The '=' (Hex 3D) character is used to avoid accidental triggering of the shutdown sequence.

DPC20xx and DPC3020

Activates the port pin with function APWR assigned to for a long period (set via the setup program). The display shows the hard-shutdown message box (defaults to "Hard Shutdown" and set via the setup program). If the hard-shutdown message is empty, no hard-shutdown message box is displayed. If no pin is assigned to the APWR port function, no hardware activity will be carried out on shutdown.

Cancel Shutdown

<CID> P U C

DPC10xx only

Cancels shutdown by doing the following things: Releases the main board's power pin and does not wait for the PC's shutdown command anymore (when the power key was pressed before). The display and the LEDs remain unchanged and pressing the power down key after this sequence has been received triggers the power down sequence again.

DPC20xx and DPC3020

Cancels shutdown by doing the following things: Deactivates the port pin with function APWR assigned to and does not wait for the PC's shutdown command anymore (when the power key was pressed before). The display and the LEDs remain unchanged and pressing the power down key after this sequence has been received triggers the power down sequence again. If no pin is assigned to the APWR port function, no hardware activity will be carried out.

Get Power State

```
<CID> P ?
```

The controller returns <ACK> `state` <ACK> where state is as follows:

`state` = Hex 00 : Power key is released, no power down sequence is running
`state` = Hex 01 : Power key is pressed, no power down sequence is running (Smart Power is off)
`state` = Hex 80 : Power key is released, power down sequence is running (Smart Power is on)
`state` = Hex 81 : Power key is pressed, power down sequence is running (Smart Power is on)

Bit 0 of `state` indicates the power key and bit 7 of state indicates a running power down sequence (only triggered when Smart Power was on at the time of the power key pressure). For more information about smart power mode see "Set Smart Power-Off Mode" on page 59.

Reset Motherboard

```
<CID> P ! =
```

DPC10xx only

Disables the watchdog and pulls the reset line of the motherboard low. LEDs and LCD display remain in the previous state, which means that if the customer should see a message and/or the LEDs should have a certain state while rebooting the PC, the corresponding sequences have to be sent by the PC before triggering the reset sequence. The `'='` (Hex 3D) character is used to avoid accidental triggering of the reset sequence.

DPC20xx and DPC3020

Disables watch dog and activates the pin assigned to the ARES function for a certain period defined via the setup program.

Set Smart Power-Off Mode

```
<CID> P S on_off
```

DPC10xx only

Enable/disable smart power functionality: When Smart Power is on (`on_off` = 1), pressing the power switch does not activate the main board's power pin directly but sends the sequence "* <ACK>" to the PC only (this sequence is sent in any case when the power switch is pressed) and sets LED 4 (power LED) to the blink mode. The PC then has to carry out the corresponding power down sequence via software. When Smart Power is off (`on_off` = 0) the power switch is directly connected to the main board's power pin.

Please note that disabling smart power has no effects on a currently running power-down sequence.

DPC20xx and DPC3020

Enable/disable smart power functionality: When Smart Power is on (`on_off` = 1), pressing the power switch does not activate the main board's power pin directly, but sends the sequence "* <ACK>" to the PC only (this sequence is sent in any case when the power switch is pressed); the PC then has to carry out the corresponding power down sequence via software. When Smart Power is off (`on_off` = 0) the power switch is directly connected to the main board's power pin.

Enabling/disabling the power switch is done via the port pin having the ASPWR function assigned to. The PC's power switch has to pull low the input pin with function APSWI assigned to.

Copyright © by demmel products, Document Date: December 2, 2009

Please note that disabling smart power has no effects on a currently running power down sequence.

Set Power-Off Notification On/Off

```
<CID> P N on_off
```

Enable/disable power switch notification: When the power switch is pressed, the sequence "* <ACK>" is sent to the PC when the notification is on (on_off = 1). When the notification is off (on_off = 0) the PC has to poll the status by the Get Power State command. Please note that even when the notification is off and the PC does not poll the power state, the shutdown sequence is started when Smart Power is on.

The startup value for the power notification can be set via the setup program.

2 3 DPC20xx and DPC3020

The PC's power switch has to pull low the input pin with function APSWI assigned to.

Data Sent By The iLCD Controller

Data Sent On Request

ACK Sequence

```
<ACK>
```

Any valid command is acknowledged by an <ACK> ('Hex 06) character. If multiple commands are sent without waiting for the <ACK> of the current command, the <ACK> character is sent on completion of the appropriate command as long as there are no buffer overflows (see below).

NACK Sequence

```
<NACK>
```

An invalid command is flagged by a <NACK> (Hex 15) character. After sending the <NACK> character the controller continues with the next valid command starting with the <CID> character.

Overflow Flag

```
<OVR>
```

The overflow flag (Hex 19) is sent when the input buffer of the panel controller is overrun. The bytes available in the input buffer are processed anyway.

EEPROM Write Error

```
<EERR>
```

<EERR> (Hex 10) is sent as a response to the Write or Erase EEPROM command, when the EEPROM section of the controller could not be programmed anymore (the EEPROM section has a suggested life time of more than 1,000,000 write cycles). If you get this message the controller usually cannot be used anymore due to excessive write accesses.

Get Error Code

```
<ACK> hb lb <ACK>
```

When receiving the "Get Last Error Code" (page 18), the controller sends a 16 bit error code as two bytes (MSB first). The error code is set by any command which can generate an error. When such a command is executed without error the error code is cleared. For a listing of all codes see Error Codes on page 66.

Firmware and Identification Information, Serial Number, iLCD Controller Name

```
<ACK> string <ACK>
```

When receiving the "Get Firmware Info" (page 18), the "Get Identification Info" (page 18), the "Get iLCD Controller Name" (page 19) or the Get Serial Number (page 18) command, the controller sends the appropriate response string.

Version String

```
<ACK> x.yy <ACK>
```

When receiving the "Get Firmware Version" command (page 18), the controller sends the version string where `x` and `y` describe the major and minor version number.

Hardware Revision

```
<ACK> x.y <ACK>
```

When receiving the "Get Hardware Revision" command (page 19), the controller sends its hardware revision string where `x` and `y` describe the major and minor version number.

Text Extent

```
<ACK> x_hb x_lb y_hb y_lb <ACK>
```

Sent as a response to the "Get Text Extent" command (page 21) or the "Get Text Message Extent" command (page 21).

Display Size

```
<ACK> width_hb width_lb height_hb height_lb <ACK>
```

Sent as a response to the "Get Display Size" command on page 23.

Backlight Mode

```
<ACK> mode <ACK>
```

Sent as a response to the "Get Backlight Mode" command on page 24.

Backlight Intensity

```
<ACK> intensity <ACK>
```

Sent as a response to the "Set Backlight Intensity" command on page 25.

Fixed Contrast/Gamma

```
<ACK> fixed <ACK>
```

Sent as a response to the "Get Fixed LCD Contrast/Gamma 3" command on page 25.

LCD Contrast

```
<ACK> contrast <ACK>
```

Sent as a response to the "Get LCD Contrast" command on page 26.

LCD Gamma Value 3

```
<ACK> gamma <ACK>
```

Sent as a response to the "Get LCD Gamma Value 3" command on page 26.

Graphics Bytes 1 2

```
<ACK> data_0 data_1 ... <ACK>
```

This sequence is sent in response to the "Read Graphics Byte 1 2" (page 31) and "Read Multiple Graphics Byte 1 2" (page 31) command.

Scan Line Data 3

```
<ACK> p0_hb p0_lb p1_hb p1_lb ... <ACK>
```

This sequence is sent in response to the "Read Scan Line 3" (page 33)

Pixel Coordinate

```
<ACK> x_hb x_lb y_hb y_lb <ACK>
```

This sequence is sent in response to the "Get Pixel Coordinate" (page 21) command.

Number of Screen Memory Positions

```
<ACK> number <ACK>
```

This sequence is sent in response to the "Get # Of Screen Memory Positions" on page 37.

EEPROM Size

```
<ACK> size_hb size_lb <ACK>
```

This sequence is sent in response to the "Get EEPROM Size 2 3" on page 56.

EEPROM Data

```
<ACK> data <ACK>
```

This sequence is sent in response to the "Read EEPROM" on page 56.

Power State

```
<ACK> state <ACK>
```

Sent as a response to the "Get Power State" (page 59) command.

Keyboard State

The keyboard state can be requested independently of the report state via a separate command (see "Get Keyboard State" on page 50).

1 DPC10xx only

The response is as follows:

```
<ACK> state_column_0 state_column_1 state_column_2 state_column_3 <ACK>
```

where each bit of `state_column_x` indicates a pressed key (row 0 = bit 0, row 1 = bit 1, ...) in the corresponding column.

2 3 DPC20xx and DPC3020

```
<ACK> state_column_0 state_column_1 ... state_column_15 <ACK>
```

where each bit of `state_column_x` indicates a pressed key (row 0 = bit 0, row 1 = bit 1, ...) in the corresponding column. Please note that columns not having assigned a KeybCol # via the setup program report all corresponding keys as not pressed.

General Inputs State

1 `<ACK> state <ACK>`
2 3 `<ACK> state_hb state_lb <ACK>`

Sent as a response to the "Get Inputs State" (page 52) command.

ADC Value

```
<ACK> value_hb value_lb <ACK>
```

Sent as a response to the "Get ADC Value" (page 52) command.

Date Value

```
<ACK> year month day weekday <ACK>
```

Sent as a response to the "Get Date 2 3" (page 53) command.

Time Value

```
<ACK> hour minute second <ACK>
```

Sent as a response to the "Get Time 2 3" (page 53) command.

Copyright © by demmel products, Document Date: December 2, 2009

Touch-Field Event

When a previously defined touch field (see at "Create/Define Touch Field" on page 45) is pressed or released or the position of pressure is changed, this event is saved in the iLCD controller and can be retrieved via the command "Retrieve Last Touch Screen Event" - see page 47 - lateron. The data returned shows up as follows:

```
<ACK> event field_idx ev_coord_x_hb ev_coord_x_lb ev_coord_y_hb
ev_coord_y_lb <ACK>
```

`event` can be as follows:

K	Touch has been pressed
k	Touch field has been released
M	Location of press has been moved
<NULL>	The Null-Byte indicates no event has occured

`field_idx` is the index of the last active touch field (0 .. max. 63) and `ev_coord_x/ ev_coord_y` is the coordinate of the event relatively to the upper left corner of the corresponding touch field. Please note, that the coordinates can become negative and the minimum / maximum coordinate values which are reported can even fall outside the physical screen dimensions by one pixel when converted to absolute coordinates.

Spontaneous Sent Data

Startup

```
# <ACK>
```

At startup, the panel controller sends the above message to the default communications port if sending the startup message is set to true via the setup program. This is also true if the controller has been rebooted by the "Reboot Panel Controller" (page 17) command.

Power Key Pressed

```
* <ACK>
```

This sequence is sent when the power key is pressed and the power-off notification is turned on independently if smart power is on or off. If smart power is on and the PC does not react within 5 seconds (can be overwritten via the setup program) a reset followed by a hard power down is started. Within this 5 seconds the `* <ACK>` sequence is repeated every second. When smart power is off, the sequence is sent only once per power key press.

Calibrate Touch Screen Done

```
T <ACK>
```

This sequence is sent after successful touch screen calibration when it was activated via the "Calibrate Touch Screen and Report" command (page 43).

Key Press/Release

When keyboard reporting is on (see at "Enable Keyboard Report" on page 50), the pressing of a key is indicated by

```
K key_char <ACK>
```

and the key release is reported as

```
k key_char <ACK>
```

where `key_char` is the character assigned to the key via the setup program.

Touch Field Press/Release

When a previously defined touch field (see at "Create/Define Touch Field" on page 45) is pressed/released, the press of a field is indicated by

```
K key_char <ACK>
```

and the release is reported as follows

```
k key_char <ACK>
```

where `key_char` is the non-zero character assigned to the touch field previously defined.

Touch Field Press/Release + Coordinate of Event 2 3

When a previously defined touch field (see at "Create/Define Touch Field" on page 45) is pressed/released, and the reporting of the coordinates is switched on via "Enable/Disable Reporting Touch-Coordinates 2 3" – see page 47, the press of a field is indicated by

```
K key_char ev_coord_x_hb ev_coord_x_lb ev_coord_y_hb ev_coord_y_lb <ACK>
```

and the release is reported as follows

```
k key_char ev_coord_x_hb ev_coord_x_lb ev_coord_y_hb ev_coord_y_lb <ACK>
```

where `key_char` is the non-zero character assigned to the touch field previously defined and `ev_coord_x/ ev_coord_y` is the coordinate where the event took place relatively to the upper left corner of the corresponding touch field. Please note, that the coordinate value may become negative when the touch field is released. The minimum / maximum coordinate value being reported can even fall outside the physical screen by one pixel when converted to absolute coordinates.

Moving Coordinate of Touch-Field Press

When a previously defined touch field (see at "Create/Define Touch Field" on page 45) is pressed and the location of the press is changed, and the reporting of the movements is switched on via "Enable/Disable Reporting Movements 2 3" – see page 47, the movement is indicated by

```
M key_char ev_coord_x_hb ev_coord_x_lb ev_coord_y_hb ev_coord_y_lb <ACK>
```

where `key_char` is the non-zero character assigned to the touch field previously defined and `ev_coord_x / ev_coord_y` is the coordinate of press relatively to the upper left corner of the corresponding touch field. Please note, that the coordinate may turn negative. The minimum / maximum coordinate values which can be reported can even fall outside the physical screen by one pixel when converted to absolute coordinates.

Copyright © by demmel products, Document Date: December 2, 2009

Error Codes

	Code	Description
E_FF_NOT_READY	201	SD-Card not ready or card missing
E_FF_NO_FILE	202	Could not find the file
E_FF_NO_PATH	203	Could not find the path
E_FF_INVALID_NAME	204	Invalid name in pathname or filename
E_FF_INVALID_DRIVE	205	Invalid drive – should not occur
E_FF_DENIED	206	Action denied Writing to file which is open for read mode File cannot created due to same name directory File cannot created due to directory table or disk full
E_FF_EXIST	207	The file is already existing
E_FF_RW_ERROR	208	The function failed due to a disk RW error or internal error
E_FF_WRITE_PROTECTED	209	Open for write and disk is write protected
E_FF_NOT_ENABLED	210	The drive has no working area – should not occur
E_FF_NO_FILESYSTEM	211	Ther is no valid FAT filesystem on the disk
E_FF_INVALID_OBJECT	212	The file object is invalid – should not occur
E_FF_MKFS_ABORTED	213	Make directory aborted due to Disk size is too small Invalid parameter Not allowable cluster size. This can occur when the number of clusters becomes around 0xFF7 or 0xFFF7
E_FF_DISKFULL	214	Disk is full
E_SDC_PARAM	230	Illegal parameter in command
E_SDC_OPEN	231	The file is already open, when issuing an open command
E_SDC_NOPEN	232	The file is not open
E_SDC_NODISK	233	No card inserted
E_SDC_EOF	234	End of file already reached, when starting read command
E_SDC_DIREND	234	End od directory reached when reading first or next directory entry
E_COMMAND	1001	Illegal command
E_X_OUT_OF_RANGE	1002	X coordinate out of range
E_Y_OUT_OF_RANGE	1003	Y coordinate out of range
E_NOT_BOOLEAN	1004	Parameter not boolean
E_VAL_OUT_OF_RANGE	1005	Value out of range
E_PARNOT_ZERO	1006	Parameter must not be zero
E_W_OUT_OF_RANGE	1007	Width out of range
E_H_OUT_OF_RANGE	1008	Height out of range
E_LCD_NOFONT	1009	Font not found
E_FIXEDTEXT	1010	Text message does not exist
E_GR_NOTFOUND	1011	Local graphic not found
E_GR_INVTYPE	1012	Invalid graphic type specified
E_GR_FRAMEIDX	1013	Invalid frame index specified
E_GR_ANIMIDX	1014	Invalid animation control index specified
E_GR_ANIMASS	1015	Animation control index not assigned
E_IO_BAUDRATE	1016	Invalid baudrate specified
E_IO_COMPORT	1017	Invalid com port specified
E_IO_LEDPAR	1018	Led number out of range
E_IO_LEDDEFINED	1019	Led output not defined
E_IO_LEDMODE	1020	Invalid led mode
E_IO_ADCCHANNEL	1021	Adc channel out of range
E_IO_ADCENABLE	1022	Adc channel not enabled
E_IO_FREQU	1023	Frequency out of range
E_IO_DUTY	1024	Duty cycle out of range
E_IO_REL_NUMBER	1025	Relais number out of range
E_IO_REL_MODE	1026	Invalid relais mode specified
E_IO_RTC_READ	1027	RTC read error

E_IO_RTC_WRITE	1028	RTC write error
E_IO_TM_HOUR	1029	Invalid hour specified
E_IO_TM_MINUTE	1030	Invalid minute specified
E_IO_TM_SECOND	1031	Invalid second specified
E_IO_DT_MONTH	1032	Invalid month specified
E_IO_DT_DAY	1033	Invalid day specified
E_IO_DT_DAYOFWEEK	1034	Invalid day of week specified
E_TCH_OUT_OF_RANGE	1035	Touch field index out of range
E_TCH_NOTCALIB	1036	Touch screen not calibrated
E_TCH_ACTIVE	1037	Touch field is not active
E_TCH_MAXFIELD	1038	Touch field index out of range
E_TCH_TEXT_NF	1039	Text message does not exist
E_MEM_OUT_OF_RANGE	1040	Memory position out of range
E_MEM_POS_EMPTY	1041	Memory position is empty
E_MEM_MAXCURSOR	1042	Cursor index out of range
E_MEM_CURSORNF	1043	Cursor memory index not assigned
E_MAC_DEPTH	1044	Macro depth nesting exceeded
E_MAC_NOT_FOUND	1045	Macro not found
E_FLSH_SETSECTOR	1046	Sector not found
E_FLSH_BLOCK	1047	NAND block out of range
E_FLSH_APPLADDR	1048	Illegal flash address specified
E_EEP_ERASE	1049	EEPROM erase error
E_EEP_MAXSIZE	1050	EEPROM address out of range

Controlling the iLCD Controller via I²C

An extra application note has been setup to learn about how to communicate with an iLCD controller via I2C. This document can be downloaded from the demmel products web site from http://www.demmel.com/download/ilcd/i2c/i2c_appnote.pdf.

Controlling the iLCD Controller via SPI

An extra application note has been setup to learn about how to communicate with an iLCD controller via SPI. This document can be downloaded from the demmel products web site from http://www.demmel.com/download/ilcd/spi_appnote.pdf.

Controlling the iLCD Controller via USB

The USB part of the iLCD controller is implemented as an HDI (Human Device Interface), this allows any iLCD controller based device to be operated via any Windows operating system with versions greater or equal Windows 98 without having to install a special driver. The HDI drivers are included in these operating systems by default, as they are used for keyboards and mice too.

A program - including the source code - for writing raw flash data files to an iLCD device can be found on http://www.demmel.com/download/ilcd/iloader.zip. This program shows how to the handle the USB port to operate with an iLCD device. It is written in standard C and can be easily adapted to other operating systems.

Note:

If your color iLCD panel has an USB port installed, the communication via this USB port has to done via the virtual COM port (the corresponding drivers will be installed during setup software installation) of the on-board USB bridge, not via "real" USB communication. "iLCD Port Selection" in the "Preferences" section of the iLCD setup software has to be set to "Serial port" in this case, the default baud rate is 115200 baud.

The baud rate of the iLCD's serial port 0 (connected to the on-board USB to serial bridge) can be changed via the setup software on by editing the setup data accordingly and rewriting the flash data to the iLCD controller then.

Sample Source Code

Two source code samples are available on the net:

Sample one is a command line tool called iLoader to download raw flash data files created with the iLCD setup software to the iLCD controller. It is written in Standard C and shows how to generally send and receive data via the serial port or USB port on the one hand and, on the other, can be integrated into an application to allow updating the flash data without having to use the setup software. The sources compile under Borland's C++ Builder 5.0 but can easily be transferred to other operating systems and/or compilers. The sample code (including the executable for Windows) can be found on the following location: http://www.demmel.com/download/ilcd/iloader.zip. Sample two shows how to operate the iLCD's I^{2}C port. This sample code contains a "driver" file, which covers all I^{2}C functions described under "Controlling the iLCD Controller via I^{2}C". This file can be used without modification in most cases for your own application. The "main" file contains a very simple example to initialize the iLCD and to show text on the screen. Both files compile under Keil C for 8051 but can be easily transferred to any other controller and/or compiler. The source code can be found on http://www.demmel.com/download/ilcd/i2c_sample_code.zip

Revision History

Date	Rev. #	Revision Details
December 2, 2009	3.41	Added commands "Set Macro Timer" and "Calibrate Touch Screen and Report"
October 5, 2009	3.4	Documents splitted, see "Scope of Document"
May 29, 2008	3.31	Added chapter "Controlling the iLCD Controller via SPI"
May 14, 2008	3.3	Added Time/Date Related Commands Added Get Fixed LCD Contrast/Gamma 3 command
September 14, 2007	3.22	Corrected hints about special touch screen related commands usable with DPC20xx and DPC3020 only
May 20, 2007	3.21	Added hint about DPC20xx and DPC3020 firmware version supporting I^{2}C
April 10, 2007	3.2	Added commands "Enable/Disable Reporting Touch-Coordinates 2 3", "Enable/Disable Reporting Movements 2 3" and "Retrieve Last Touch Screen Event 2 3". Chapters "Important Chapters to Read" and "Syntax Used in Setup Program" added.
February 10, 2007	3.1	Added command "Set Screen Orientation 3"
January 25, 2007	3.0	Added commands for DPC10xx and DPC20xx iLCD controller family
February 14, 2006	2.11	Added "shortcut"-behavior for "Draw Touch Field Text"
August 01, 2005	2.1	Added section "Controlling the iLCD Controller via I^{2}C" Added section "Controlling the iLCD Controller via USB"
April 05, 2005	2.0	Added Touch Screen Related Commands Added command: Increment/Decrement Column Address Added command: Increment/Decrement Row Address Added command: Allow Keyboard/Touch Macros To Start
August 13, 2004	1.03	Added startup values for Pulse Width Modulation (PWM) Related Commands
June 15, 2004	1.02	Added some general info for Command Structure. Startup sequence is now sent on demand. Fixed error in description of bits in "Set Text Alignment". Screen Memory Related Commands now take care of the currently set inverse mode. Added PWM documentation due to iLCD hardware version 3.0
April 18, 2004	1.01	Startup sequence is not sent anymore – removed. Added "Get Chip Name" and "Get Hardware Revision" command.
March 8, 2004	1.0	First Issue.

If you find any errors in this document, please contact demmel products at support@demmel.com

Copyright © by demmel products, Document Date: December 2, 2009