Contents

0 (0x00) Ping

Used to verify communication with the module. The module returns the data back to the host unaltered.

Command Packet:

type:	0 (0x00)
data_length:	0 to 124
data[]:	any arbitrary data

Successful Return Packet:

type:	64 (0x00 | 0x40)
data_length:	(identical to received packet)
data[]:	(identical to received packet)

1 (0x01) Module Information

Query module type, and versioning information.

Read Module Version

Command Packet:

type:	1 (0x01)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	65 (0x01 | 0x40)
data_length:	16 to 32
data[]:	"CFA039A0-N-V:hX.X,fX.X" formatted ASCII string hX.X is the hardware version fX.X is the firmware version

Read Module Serial Number

Command Packet:

type:	1 (0x01)
data_length:	1
data[0]:	1

Successful Return Packet:

type:	65 (0x01 | 0x40)
data_length:	0 to 32
data[]:	module serial number ASCII string

Read Bootloader Version

Command Packet:

type:	1 (0x01)
data_length:	1
data[0]:	2

Successful Return Packet:

type:	65 (0x01 | 0x40)
data_length:	0 to 16
data[]:	"X.X" formatted ASCII bootloader version string

Read Firmware Build Date

Command Packet:

type:	1 (0x01)
data_length:	1
data[0]:	3

Successful Return Packet:

type:	65 (0x01 | 0x40)
data_length:	15
d ata[]:	"YYYYMMDD-HHMMSS" formatted ASCII date/time string

2 (0x02) Write User Flash Area

The module reserves 124 bytes of nonvolatile memory for arbitrary use by the host. This memory can be used to store a serial number, IP address, gateway address, netmask, or any other data required. The reply packet is returned to the host when the command has completed.

The data is saved in a file on the on-board flash filesystem, named “a:\userflash.data”. If the file cannot be written to, an error packet will be returned.

Note: if the file is removed, or the on-board flash filesystem is formatted, the data will be lost.

Command Packet:

type:	2 (0x02)
data_length:	1 to 124
data[]:	arbitrary user data to be stored in the nonvolatile memory

Successful Return Packet:

type:	66 (0x02 | 0x40)
data_length:	0
data[]:	N/A

3 (0x03) Read User Flash Area

This command read the user flash data as previously saved by command 2.
See command 2 for more details.

This command first attempts to open data file “b:\userflash.data” on the microSD card. If it exists it copies the file to the on-board flash, then renames the microSD file to “userflash.data.bak”. It will then return the data present in the file.

Command Packet:

type:	3 (0x03)
data_length:	1
data[0]:	number of bytes of data to be returned (1 to 124)

Successful Return Packet:

type:	67 (0x03 | 0x40)
data_length:	number of bytes specified in command
data[]:	user data saved previously by command 2

4 (0x04) Store Current State as Boot State

This command stores the module current state, which is then loaded on module power-on or when the module is reset (see command 5).
The return packet is sent to the host when the saving process has been completed.

The module state is saved in a file on the on-board flash filesystem, named “a:\settings.ini”.
If the file cannot be written to, an error packet will be returned.

Note that if the module state includes many graphical objects (images, fonts, videos, etc), loading of the saved state on module power-on, reset, etc will require more time (possibly seconds).

This command saves all module state information **except** these items: - current state of graphics objects (the individual commands set the stored starting states) - debugging output enabled by command 62

Important Note:
Do not attempt to edit the “settings.ini” file manually. It uses very specific formatting that is easily broken and will lead to undefined module behavior.

Command Packet:

type:	4 (0x04)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	68 (0x04 | 0x40)
data_length:	0
data[]:	N/A

5 (0x05) Module Reset/Restart

This command provides various module (and host, if using ATX functions) power, reset and restart functions as detailed below.

See command 4 for details on saving module settings.

Note:
Any time the module loads it’s saved settings (module power-on, module reset, load-settings command, etc), it will first check for the presence of file “b:\settings.ini” on the microSD card. If it exists it will copy the file to the on-board flash filesystem, then renames the microSD file to “settings.ini.bak”.
It will then load the settings from “a:\settings.ini”.

Load Saved Settings

This command loads the module settings from the saved settings file, then sends the reply packet to the host. If the saved settings file does not exist, the module will return an error packet.

Command Packet:

type:	5 (0x05)
data_length:	3

data[0]:	8
data[1]:	18
data[2]:	99

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

Restart Host

When issued this command, the module will reply with a return packet, then pull the ATX reset GPIO pin high/low (depending on the ATX polarity sense setting), while displaying a “SYSTEM RESTART” message on the LCD, then module will load it’s saved settings after 1 second.
This command will also restart any attached CFA-FBSCAB modules to the state saved in their non-volatile memory.

Command Packet:

type:	5 (0x05)
data_length:	3
data[0]:	12
data[1]:	28
data[2]:	97

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

Power-Off Host

When issued this command, the module will reply with a return packet, then pull the ATX power GPIO pin high/low (depending on the ATX polarity sense setting), while displaying a “SYSTEM POWER-OFF” message on the LCD, then the module will load it’s saved settings.
This command will also restart any attached CFA-FBSCAB modules to the state saved in their non-volatile memory.

Command Packet:

type:	5 (0x05)
data_length:	3
data[0]:	3
data[1]:	11
data[2]:	95

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

Module Restart

Performs a restart of the module. This command also restarts any attached CFA-FBSCAB modules to the state saved in their non-volatile memory.
The module will return the acknowledge packet immediately, then restart itself.

The module may not respond to new command packets for a few seconds. The length of time depends on the graphics items that need to be loaded.

If using the USB (virtual COM port) interface, this command will cause the module to disconnect and then re-connect (re-enumerate). Software running on the host may need to close, and re-open the virtual COM port for communications to resume.

Command Packet:

type:	5 (0x05)
data_length:	3
data[0]:	8
data[1]:	25
data[2]:	48

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

Backwards Compatible: Yes (CFA635, CFA735, CFA835).

Restore Defaults & Reset

Resets the system boot state to “factory defaults” by deleting the “settings.ini” file on both the microSD card filesystem, and on-board flash filesystem. After removing both files, the module restarts itself.

This command will also reset any attached CFA-FBSCAB to the state saved in their non-volatile memory. This option does not affect any other files present on the microSD card filesystem, on-board flash filesystem, or the user flash values set by command 2.

If using the USB (virtual COM port) interface, this command will cause the module to disconnect and then re-connect (re-enumerate). Software running on the host may need to close, and re-open the virtual COM port for communications to resume.

Command Packet:

type:	5 (0x05)
data_length:	3
data[0]:	10
data[1]:	8
data[2]:	98

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

Restart into Bootloader Mode

Restarts the module, and puts it into Bootloader mode.
This command may be useful for field firmware updates.

Command Packet:

type:	5 (0x05)
data_length:	3
data[0]:	88
data[1]:	207
data[2]:	5

Successful Return Packet:

type:	69 (0x05 | 0x40)
data_length:	0
data[]:	N/A

6 (0x06) Clear Display

Deletes all existing graphical objects, and clears the backwards-compatible CFA635/CFA735/CFA835 legacy text/graphics layer.

Command Packet:

type:	6 (0x06)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	70 (0x06 | 0x40)
data_length:	0
data[]:	N/A

9 (0x09) Legacy Special Character Bitmaps

Sets the bitmap for one of the 8 special characters in the CGRAM to be used with command 31.

Note:
This command is included for backwards compatibility only. It is not recommended for use in new designs.

Set Custom Character Data

Command Packet:

type:	9 (0x09)
data_length:	9
data[0]:	character index (0 to 7)
data[1-8]:	character data

Successful Return Packet:

type:	73 (0x09 | 0x40)
data_length:	0
data[]:	N/A

Read Custom Character Data

Command Packet:

type:	9 (0x09)
data_length:	1
data[0]:	character index (0 to 7)

Successful Return Packet:

type:	73 (0x09 | 0x40)
data_length:	8
data[0-7]:	character data

11 (0x0B) Legacy Cursor Position

Sets/reads the current cursor position on the backwards-compatible CFA635/CFA735/CFA835 legacy text layer.
To set the cursor visibility and style, see command 12.

Note:
This command is included for backwards-compatibility only. It is not recommended for use in new designs.

Set Cursor Location

Command Packet:

type:	11 (0x0B)
data_length:	2
data[0]:	column position
data[1]:	row position

Successful Return Packet:

type:	75 (0x0B | 0x40)
data_length:	0
data[]:	N/A

Read Cursor Location

Command Packet:

type:	11 (0x0B)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	75 (0x0B | 0x40)
data_length:	2
data[0]:	column position
data[1]:	row position

12 (0x0C) Legacy Cursor Style

Sets/reads the current cursor style on the backwards-compatible CFA635/CFA735/CFA835 legacy text layer.
To set the cursor position, see command 11.

Note:
Command is backwards compatible with CFA635, CFA735, CFA835, but cursor style may be different de-

pending on the model.
This command is included for backwards-compatibility only. It is not recommended for use in new designs.

Set Cursor Style

Command Packet:

type:	12 (0x0C)
data_length:	1
data[0]:	style: 0 = hidden (no) cursor 1 = blinking block cursor 2 = underscore cursor 3 = blinking block with underscore 4 = inverting blinking block

Successful Return Packet:

type:	76 (0x0C | 0x40)
data_length:	0
data[]:	N/A

Read Cursor Style

Command Packet:

type:	12 (0x0C)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	76 (0x0C | 0x40)
data_length:	2
data[0]:	current style (as above)

13 (0x0D) Display Contrast

This command is included for backwards compatibility only. It has no effect.
The display panel used on this module has a pre-set contrast adjustment.

Set Display Contrast

Command Packet:

type:	13 (0x0D)
data_length:	1
data[0]:	contrast setting (0 to 255 valid)

Successful Return Packet:

type:	77 (0x0D | 0x40)
data_length:	0
data[]:	N/A

Read Display Contrast

Command Packet:

type:	13 (0x0D)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	77 (0x0D | 0x40)
data_length:	1
data[0]:	127 (always reports this value)

14 (0x0E) Lighting

Module lighting control related commands. See individual command details below.

Set Display Backlight Brightness

Sets display backlight brightness.

Command Packet:

type:	14 (0x0E)
data_length:	1
data[0]:	backlight brightness (0 to 100 valid)

Successful Return Packet:

type:	78 (0x0E | 0x40)
data_length:	0
data[]:	N/A

Set Display Backlight & Keypad Brightness

Sets display backlight and keypad brightness.
For backwards compatibility this command has no keypad color control. The keypad is set to white color of the specifed brightness.

Command Packet:

type:	14 (0x0E)
data_length:	2
data[0]:	backlight brightness (0 to 100 valid)
data[1]:	keypad brightness (0 to 100 valid)

Successful Return Packet:

type:	78 (0x0E | 0x40)
data_length:	0
data[]:	N/A

Read Display Backlight & Keypad Brightness

Read the current backlight and keypad brightness levels.
To maintain backwards compatibility, the returned value for keypad brightness is the value of the brightest individual keypad RGB LED.

Command Packet:

type:	14 (0x0E)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	78 (0x0E | 0x40)
data_length:	2
data[0]:	current backlight brightness (0 to 100)
data[1]:	current keypad brightness (0 to 100)

Set Keypad and Indicator LED Color / Brightness

Set color/brightness of individual or multiple keypad and indicator LEDs.
Bits 0 to 3 of indicator leds are the on-board LEDs ordered top to bottom respectivley. The remaining high 4 bits can be used to control 4 more LEDs attached to the LED expansion header H3.

More than one LED can be updated at once by specifiying multiple LED bits. For example \002\005\255\000\255 would set the 2nd indicator LED, 1st & 3rd keypad leds to bright purple, and leave all other LEDs at thier previous values.

Command Packet:

type:	14 (0x0E)
data_length:	5
data[0]:	bitmask of indicator leds to change: bit0 = first (top) indicator led bit1 = second indicator led bit2 = third indicator led bit3 = fourth (bottom) indicator led bit4-7 = external LEDs connected on expansion header H3
data[1]:	bitmask of keypad leds to change: bit0 = up key bit1 = enter key bit2 = cancel key bit3 = left key bit4 = right key bit5 = down key
data[2]:	red value (0 to 255)
data[3]:	green value (0 to 255)
data[4]:	blue value (0 to 255)

Successful Return Packet:

type:	78 (0x0E | 0x40)
data_length:	0
data[]:	N/A

Lighting Idle Dimming

The module can automatically dim lighting after specified time(s) of inactivity. The user has the option of using one or to steps (levels) of dimming.

The lighting dimming progression is: Normal -> Level1 -> Level2. If only one step of dimming is needed, set level 2 timeout to a value of 0. Level 2 brightness must be lower (darker) than level 1.

Command Packet:

type:	14 (0x0E)
data_length:	6
data[0]:	dimming effects: bit0 = lcd backlight bit1 = keypad leds bit2 = indicator leds
data[1]:	activity monitoring: bit0 = keypad presses bit1 = touchscreen bit2 = incoming command packets
data[2]:	level-1 dim timeout seconds (0=disabled)
data[3]:	level-2 dim timeout seconds (0=disabled)
data[4]:	level-1 dimming percentage (0 to 100 valid, 100 = disabled)
data[5]:	level-2 dimming percentage (0 to 100 valid, 100 = disabled)

Successful Return Packet:

type:	78 (0x0E | 0x40)
data_length:	0
data[]:	N/A

23 (0x17) Keypad Reporting

By default, the module reports any key event to the host. This command allows the key events to be enabled or disabled on an individual basis. This command can also be used to read the current key reporting masks.

Set Keypad Reporting

Command Packet:

type:	23 (0x17)
data_length:	2
data[0]:	keypad press flags bit0 = up key bit1 = enter key bit2 = cancel key bit3 = left key bit4 = right key bit5 = down key
data[1]:	keypad release flags (bit flags same as above)

Successful Return Packet:

type:	87 (0x17 | 0x40)
data_length:	0
data[]:	N/A

Read Keypad Reporting Settings

Command Packet:

type:	23 (0x17)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	87 (0x17 | 0x40)
data_length:	2
data[0]:	current keypad press flags (bit flags same as above)
data[1]:	keypad release flags (bit flags same as above)

Keypad Report Packet

Depending on the set keypad press/release flags, the following report packets are sent to the host.

Note:
Individual communications interfaces can be configured to send or not send report packets. See command 33 for details.

Report Packet:

type:	128 (0x00 | 0x80)
data_length:	1
data[0]:	keypad activity code: 0 = none 1 = up key pressed 2 = down key pressed 3 = left key pressed 4 = right key pressed 5 = enter key pressed 6 = cancel key pressed 7 = up key released 8 = down key released 9 = left key released 10 = right key released 11 = enter key released 12 = cancel key released

24 (0x18) Keypad Poll State

Use this command to poll the current, and recent state of the keypad keys.
This command is independent of the key reporting masks set by command 23. All keys are always visible to this command.

Command Packet:

type:	24 (0x18)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	88 (0x18 | 0x40)
data_length:	3
data[0]:	keys currently pressed: bit0 = up key bit1 = enter key bit2 = cancel key bit3 = left key bit4 = right key bit5 = down key
data[1]:	keys pressed since last polled:

(bit flags same as above)
data[2]: keys released since last polled:
(bit flags same as above)

25 (0x19) Touchscreen Reporting

This command configures touchscreen activity reporting. If enabled, this command will always report absolute touch locations, independent of any graphical objects that are displayed.

Individual graphical objects also have the option of having touch reporting enabled.
See command 40, subcommand 12, and each of the graphical object creation commands for more information.

Note:
This command is available on all versions of the module, but will only send touch report packets when the module is fitted with an optional touchscreen.

Set Reporting Settings

Command Packet:

type: 25 (0x19)
data_length: 1
data[0]: touch reporting option:
0 = off
1 = report single touch & release (including position)
2 = report single touch & release and all position updates
3 = report multiple touch tracking (all updates)

Successful Return Packet:

type: 89 (0x19 | 0x40)
data_length: 0
data[]: N/A

Read Reporting Settings

Command Packet:

type: 25 (0x19)
data_length: 0
data[]: N/A

Successful Return Packet:

type: 89 (0x19 | 0x40)
data_length: 1
data[0]: touch reporting option:
0 = off
1 = report single touch & release (including position)
2 = report single touch & release and all position updates
3 = report multiple touch tracking (all updates)

Touchscreen Report Packet

Note:
Individual communications interfaces can be configured to send or not send report packets. See command 33 for details.

Report Packet:

type:	131 (0x03 | 0x80)
data_length:	1 + (touch-count * 6)
data[0]:	number of touch points (touch-count)
data[1]:	track id of touch point
data[2]:	status flags: bit0 = released bit1 = pressed bit2 = held/dragged
data[3]:	touch x pixel position (16bit, unsigned, least significant byte)
data[4]:	touch x pixel position (16bit, unsigned, most significant byte)
data[5]:	touch y pixel position (16bit, unsigned, least significant byte)
data[6]:	touch y pixel position (16bit, unsigned, most significant byte)
data[7+]:	data[1-6] repeated X times, depending on number of touch-count

28 (0x1C) ATX Functionality

With ATX features enabled, the module can control the functions of the power and restart switches in a standard ATX-compatible system. User/host Control is provided via the modules keypad buttons and/or packet commands (see command 5 for details).

For details on the wiring connections required for this feature, see the ATX Power Supply and Control Connections section.

MOVE THIS TEXT TO ABOVE MENTIONED SECTION

The RESET (GPIO[3]) and POWER CONTROL (GPIO[2]) lines on the module are normally high-impedance. Electrically, they appear to be disconnected or floating. When the CFA835 asserts the RESTART or POWER CONTROL lines, they are momentarily driven high or low (as determined by the RESTART_INVERT and POWER_INVERT bits, detailed below). To end the power or restart pulse, the CFA835 changes the lines back to high-impedance.

Note:
The H1 GPIO pins used for ATX control must be configured to their default drive mode in order for the ATX functions to work correctly. See command 34.
The SPI interface on H1 must also be disabled. See command 33.

Function: Keypad Restart

If ATX Power Sense IO pin (GPIO[1]) is high, holding the green check key for 4 seconds will pulse the ATX Reset Control IO pin (GPIO[3]) pin for 1 second. During the 1-second pulse, the module will show “SYSTEM RESTART”, and then load it’s saved settings.

Function: Keypad Power On

If the ATX Power Sense IO pin (GPIO[1]) is low, pressing the green check key for 0.25 seconds will pulse ATX Power Control IO pin (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. During this time, the module will show “SYSTEM POWER ON”, and then load it’s saved settings.

Function: Keypad Power-Off

If ATX Power Sense IO pin (GPIO[1]) is high, holding the red X key for 4 seconds will pulse the ATX Power Control IO pin (GPIO[2]) for the duration specified by in data[1] or the default of 1 second. If the user continues to hold the power key down, then the module will continue to drive the line for a maximum of 5 additional seconds. During this time, the module will show “SYSTEM POWER OFF”.

Function: Module / Lighting Mimics Host Power

If the “module mimic host power” option flag is set, the module will turn off the display and keypad backlights when the ATX Power Sense IO pin (GPIO[1]) is low (indicator LEDs will remain on at previous setting).
If the “lighting mimics host power” option flag is also set, the indicator LEDs will also be turned off. The module will still be active (powered by the host stand-by power VSB to H1) and monitoring the keypad for a power-on keystroke.
When the ATX Power Sense IO pin (GPIO[1]) returns to high, the module will load it’s saved settings, and resume normal operation.
If using a FBSCAB to control fans, if +12v remains active (which would not be expected, since the host is “off”), the fans will remain on at their previous settings.

Set ATX Functionality

Command Packet:

type:	28 (0x1C)
data_length:	1 or 2

data[0]: option flags of enabled functions:
- bit0 = auto polarity (detects polarity for restart and power)
- bit1 = restart invert (restart pin drives high instead of low)
- bit2 = power invert (power pin drives high instead of low)
- bit3 = lighting mimics host power (turn off the LEDs also if host power sense is low)
- bit4 = module mimic host power (turn off the display if the host power sense is low)
- bit5 = keypad restart enabled
- bit6 = keypad power-on enabled
- bit7 = keypad power-off enabled

data[1]: length of power on & off pulses in 1/32 second increments
- 1 = 1/32 second
- 2 = 1/16 second
- 16 = 1/2 second
- ...
- 254 = 7.9 second
- 255 = hold until power-on sense change or 1 second, whichever is shorter

Note:
Power and restart invert (bit1, bit2) flags are ignored if auto-polarity (bit0) is enabled.

Successful Return Packet:

type: 92 (0x1C | 0x40)
data_length: 0
data[]: N/A

Read ATX Functionality

Command Packet:

type: 28 (0x1C)
data_length: 0
data[0]: N/A

Successful Return Packet:

type: 92 (0x1C | 0x40)
data_length: 2
data[0]: option flags of enabled functions (as above)
data[1]: length of power on & off pulses in 1/32 second increments

29 (0x1D) Watchdog

The module can act as a watchdog for the host system if the ATX functionality of the module is also in use (see command 28).
When enabled, if the host does not re-send this command to the module within the supplied time period, the module will perform a ATX reset of the host and restart itself as if command 5 restart command was issued.

This command is not active by default after module power-on/reset, and it's settings are not saved by command 4. If the host is started or reset by the ATX functions of the module, the watchdog function will not become active until the host's software re-issues the this command to the module.

To disable the watchdog function after it has been enabled, set the time-out to a value of 0.

Note: The GPIO pins used for ATX control must not be configured as user GPIO, or used for the SPI interface. They must be configured to their default drive mode in order for the ATX functions to work correctly. These settings are factory default, but may be changed by the user. See the note under command 28 and command 34.

Write/Reset Watchdog Timeout

Command Packet:

type:	29 (0x1D)
data_length:	1
data[0]:	watchdog timeout in seconds (0=disabled)

Successful Return Packet:

type:	93 (0x1D | 0x40)
data_length:	0
data[]:	N/A

Read Watchdog Timeout

Command Packet:

type:	29 (0x1D)
data_length:	0
data[]:	N/A

Successful Return Packet:

type:	93 (0x1D | 0x40)
data_length:	1
data[0]:	watchdog timeout in seconds (0=disabled)

31 (0x1F) Legacy Write Text to the Display

This command places text and special characters at any position on the the backwards-compatible CFA635/CFA735/CFA835 legacy text layer.

Note:
This command is included for backwards compatibility only. It is not recommended for use in new designs. The custom font options of the CFA835 are not available on this module.

Command Packet:

type:	31 (0x1F)
data_length:	3 to 22
data[0]:	column position (0 to 19 valid)

data[1]: row position (0 to 3 valid)
data[2-21]: text to place on the display

Successful Return Packet:

type: 95 (0x1F | 0x40)
data_length: 0
data[]: N/A

32 (0x20) Legacy Read Text from the Display

This command allows the host to read text and special characters that back from the backwards-compatible CFA635/CFA735/CFA835 legacy text layer.

Note:
This command is included for backwards compatibility only. It is not recommended for use in new designs.

Command Packet:

type: 32 (0x20)
data_length: 3
data[0]: column position (0 to 19 valid)
data[1]: row position (0 to 3 valid)
data[2]: length of text to read

Successful Return Packet:

type: 96 (0x20 | 0x40)
data_length: 1 to 20
data[]: read text

33 (0x21) Interface Options

This command sets or reads module communication interface options.
The module has four interfaces available for communications:

- USB 2.0 Full Speed (CDC-ACM virtual COM port)
- Serial (logic level, or “full-swing” RS232 with attached sub-board option)
- I2C (slave only)
- SPI (slave only)

All interfaces may be used at any time (when enabled with this command) including being used simultaneously. See the Connection Information section for details on physical/electrical connections on the H1 connector. Some pins on H1 used for Serial / I2C / SPI interfaces are shared with other module functions (for example, GPIO, ATX power control, ADC). When an interface is enabled, it will override any other H1 pin use. For example, if the SPI interface is enabled, ATX power control will no longer be available.

The following option flags are applicable to all the interface commands below:

bit0 = enable interface
bit1 = enable command interpreter on this interface
bit2 = enable report packets on this interface
bit3 = enable error packets for commands on this interface
bit4 = enable error packets for commands on any interface

When option bit0 and bit1 are enabled, the module will accept packets on the interface. Options bit2, bit3, bit4 only have an effect when bit0 and bit1 are enabled. Command return packets are only sent to the originating interface when bit2 is enabled.
The bit4 option is useful for debugging interface problems on other enabled interfaces.

Set Serial Interface Options

Command Packet:

type:	33 (0x21)
data_length:	3
data[0]:	0 (Serial)
data[1]:	option flags (as above)
data[2]:	baud rate: 0 = 19200 1 = 115200 2 = 9600

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	0
data[]:	N/A

Read Serial Interface Options

Command Packet:

type:	33 (0x21)
data_length:	1
data[0]:	0 (Serial)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	3
data[0]:	0 (Serial)
data[1]:	option flags (as above)
data[2]:	baud rate (as above)

Set USB Interface Options

Note: The USB interface cannot be fully disabled. Option bit0 has no effect.

Command Packet:

type:	33 (0x21)
data_length:	2
data[0]:	1 (USB)
data[1]:	option flags (as above)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	0
data[]:	N/A

Read USB Interface Options

Command Packet:

type:	33 (0x21)
data_length:	1
data[0]:	1 (USB)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	2
data[0]:	1 (USB)
data[1]:	option flags (as above)

Set SPI Interface Options

Command Packet:

type:	33 (0x21)
data_length:	4
data[0]:	2 (SPI)
data[1]:	option flags (as above)
data[2]:	SPI mode settings bit0 = SPI CPOL (0 = 1st edge, 1 = 2nd edge) bit1 = SPI CPHA (0 = polarity low, 1 = polarity high) bit2 = bit first (0 = MSB first, 1 = LSB first)
data[3]:	0 (reserved for future use)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	0
data[]:	N/A

Read SPI Interface Options

Command Packet:

type:	33 (0x21)
data_length:	1
data[0]:	2 (SPI)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	4
data[0]:	2 (SPI)
data[1]:	option flags (as above)
data[2]:	SPI mode settings
data[3]:	0 (reserved for future use)

Set I2C Interface Options

Command Packet:

type:	33 (0x21)
data_length:	4
data[0]:	3 (I2C)
data[1]:	option flags (as above)
data[2]:	I2C address (0x00 to 0x7F)

data[3]:	I2C bus speed 0 = 100khz 1 = 400khz

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	0
data[]:	N/A

Read I2C Interface Options

Command Packet:

type:	33 (0x21)
data_length:	1
data[0]:	3 (I2C)

Successful Return Packet:

type:	97 (0x21 | 0x40)
data_length:	4
data[0]:	3 (I2C)
data[1]:	option flags (as above)
data[2]:	I2C address
data[3]:	I2C bus speed

34 (0x22) GPIO Pin Configuration

This command configures, and set/reads the state of the GPIO pins on H1 of the module.

The module has thirteen pins available on the H1 connector for user-definable general-purpose input/output (GPIO), communications interfaces, analog to digital converter (ADC), and ATX power control functions. See section Connection Information for more information on H1 pinout and pin functions.

The module continuously samples the GPIO pins states at 200Hz. It also keeps track of rising or falling edges since the last host query. This means that the host is not forced to query the module quickly in order to detect short events.

Two ADC input pins are also provided on H1 pins 5 & 6. When the ADC is enabled, the sampled ADC values are averaged between the host reading the values using this command. The averaged value is multiplied by 16 to increase value accuracy over long sample periods. The minimum and maximum ADC values are also tracked between the host reading values using this command.
The ADC has 12-bit resolution, and uses a 3.3V reference voltage (min=3.27v max=3.39v).

The GPIO pins may also be used for ATX power control through the H1 connector using the WR-PWR-Y25 ATX power switch cable (or a custom harness). The ATX power control features can only be enabled using command 28 if the relevent GPIO pins (see below) are set to the unused/default function mode.

Note:
The LED control features of this command are limited, and included for backwards compatibility only. It is recommended that new designs use command 14 for LED control.

H1 Connector GPIO Index

GPIO Index	GPIO / Pin Name	H1 Pin	Default Function
0	GPIO[0]	Pin 11	Unused (Hi-Z)
1	GPIO[1]	Pin 12	ATX Host Power Sense
2	GPIO[2]	Pin 9	ATX Host Power Control
3	GPIO[3]	Pin 10	ATX Host Reset Control
4	GPIO[4]	Pin 13	Unused (Hi-Z)
5	LED 3 Green	N/A	LED Off
6	LED 3 Red	N/A	LED Off
7	LED 2 Green	N/A	LED Off
8	LED 2 Red	N/A	LED Off
9	LED 1 Green	N/A	LED Off
10	LED 1 Red	N/A	LED Off
11	LED 0 Green	N/A	LED 100% On
12	LED 0 Red	N/A	LED Off
13	GPIO[5]	Pin 5	ADC 0 Input
14	GPIO[6]	Pin 6	ADC 1 Input
15	GPIO[7]	Pin 1	Serial TX
16	GPIO[8]	Pin 2	Serial RX
17	GPIO[9]	Pin 3	Unused (Hi-Z)
18	GPIO[10]	Pin 4	Unused (Hi-Z)
19	GPIO[11]	Pin 7	Unused (Hi-Z)

GPIO Index	GPIO / Pin Name	H1 Pin	Default Function
20	GPIO[12]	Pin 8	Unused (Hi-Z)

Set Pin Value & Configuration

Command Packet:

type: 34 (0x22)
data_length: 2 or 3
data[0]: index of GPIO to modify (0-20 valid, see table above)
data[1]: pin output value (behavior depends on drive mode) (0-100 valid)
- 0 = output set to low
- 1-99 = output PWM duty cycle percentage
- 100 = output set to high

data[2]: pin function and drive mode:
- bit0-2 = drive mode:
  - 000 = strong drive up, resistive pull down
  - 001 = strong drive up, strong drive down
  - 010 = hi-z (use for input)
  - 011 = resistive pull up, strong drive down
  - 100 = strong drive up, hi-z drive down
  - 101 = strong drive up, no drive down (hi-z)
  - 110 = reserved, do not use (returns error packet)
  - 111 = no drive up (hi-z), strong drive down
- bit3 = function:
  - 0 = pin unused / default mode (see table above)
  - 1 = pin under user control
- bit4-7 = reserved (must be 0)

Successful Return Packet:

type: 98 (0x22 | 0x40)
data_length: 0
data[]: N/A

Read Pin Value & Configuration

Command Packet:

type: 34 (0x22)
data_length: 1
data[0]: index of GPIO to read (0-20 valid, see table above)

Successful Return Packet:

type: 98 (0x22 | 0x40)
data_length: 4
data[0]: index of the GPIO read
data[1]: pin state:
- bit0 = last sampled pin state
- bit1 = set if one or more falling edges have been detected on this pin since this command was last sent
- bit2 = set if one or more rising edges have been detected on this pin since this command was last sent
- bit3-7 = reserved

data[2]: pin output value (0 to 100)
data[3]: pin function and drive mode:
- bit0-2 = drive mode:
  - 000 = strong drive up, resistive pull down

001 = strong drive up, strong drive down
010 = hi-z (use for input)
011 = resistive pull up, strong drive down
100 = strong drive up, hi-z drive down
101 = strong drive up, no drive down (hi-z)
110 = reserved, do not use (returns error packet)
111 = no drive up (hi-z), strong drive down

bit3 = function:
0 = pin unused/default (see table above)
1 = pin under user control

bit4-7 = reserved (must be 0)

Read Pin ADC Values

This command only applies to GPIO Index 13 and 14 (H1 GPIO pins 5 and 6 respectively).
The pin must be set as unused/default function for the following ADC return packet to be sent by the module. If the pin is not set as unused/default function, the above “GPIO Read” formatted return packet will be sent to the host.

Command Packet:

type:	34 (0x22)
data_length:	1
data[0]:	index of GPIO ADC to read (13-14 valid)

Successful Return Packet:

type:	98 (0x22 | 0x40)
data_length:	7
data[0]:	index of GPIO ADC
data[1]:	average of samples since last read * 16 (16bit, unsigned, low-byte)
data[2]:	average of samples since last read * 16 (16bit, unsigned, high-byte)
data[3]:	minimum sample value since last read (16bit, unsigned, low-byte)
data[4]:	minimum sample value since last read (16bit, unsigned, high-byte)
data[5]:	maximum sample value since last read (16bit, unsigned, low-byte)
data[6]:	maximum sample value since last read (16bit, unsigned, high-byte)

37 (0x25): CFA-FBSCAB Command Group

The module supports fans, temperature sensors, and additional GPIOs through the addition of one or more CFA-FBSCABs.
This command group contains all of the subcommands necessary to interact with the attached FBSCABs including reading and writing from the FBSCABs fans, temperature sensors, and GPIO pins. As many as 32x FBSCABs can be daisy-chained using WR-EXT-Y37 communication cables. The combination of the module plus one or more CFA-FBSCABs can be used as part of an active fan cooling system. The fans can be slowed down to reduce noise when a system is idle or when the ambient temperature is low. The fans speed up when the system is under heavy load or the ambient temperature is high.
The module can automatically control the speed of the fans by use of the auto fan control feature, or the host system can control the attached fans power using subcommand 1.

See the subcommands below for detailed information on FBSCAB operations.

Subcommand 0 - Read CFA-FBSCAB Information

This subcommand returns the quantity of FBSCABs detected by the module or the serial number of a specified FBSCAB.

Query Number of FBSCABs

Command Packet:

type:	37 (0x25)
data_length:	1
data[0]:	0 (subcommand number)

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	2
data[0]:	0 (subcommand number)
data[1]:	number of deteced attached FBSCABs

Query FBSCAB Serial Number

Command Packet:

type:	37 (0x25)
data_length:	2
data[0]:	0 (subcommand number)
data[1]:	FBSCAB index to query

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	18
data[0]:	0 (subcommand number)
data[1]:	index of queried FBSCAB
data[2-17]:	FBSCAB serial ASCII string

Subcommand 1 - Fan Settings

This subcommand configures or reads the power settings of the fan connectors on the specified FBSCAB.

Fan power is controlled by PWM switching the fan's power supply at approximately 18Hz.
A fan power control fail-safe system is provided, and controlled by this subcommand. If the fail-safe bit for a fan is enabled and the fan power level is not updated by the host system using this subcommand within the time-out period, the fans with the fail-safe bit enabled will have the power level set to 100% until this subcommand packet is received.

This subcommand also allows setting a variable-length delay (glitch delay) after the fan has been turned on before the CFA835 will recognize transitions on the tachometer line. Some fans require a longer delay for the module to reliably read the tachometer output. The delay is specified in counts, each count being nominally 552.5 µS long (1/100 of one period of the 18Hz PWM repetition rate). In practice, most fans will not need the delay to be changed from the default length of 1 count. If a fan's tachometer output is not stable when its PWM setting is other than 100%, simply increase the delay until the reading is stable.
Typically, start at a delay count of 50 or 100, reduce it until the problem reappears, and then slightly increase the delay count to give it some margin.
Setting the glitch delay to higher values will make the fan tachometer monitoring slightly more intrusive at low power settings. Also, the higher values will increase the lowest speed that a fan with tachometer reporting enabled will "seek" at "0%" power setting.

Set Fan Power, Fail-Safe and Glitch

Command Packet:

type:	37 (0x25)
data_length:	6 or 8 or 12
data[0]:	1 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	power level for FAN 1 (0-100 valid)
data[3]:	power level for FAN 2 (0-100 valid)
data[4]:	power level for FAN 3 (0-100 valid)
data[5]:	power level for FAN 4 (0-100 valid)
data[6]:	fail-safe enabled for these fans bitmask
data[7]:	fan power update must happen within this many 1/8 second periods
data[8]:	glitch delay for FAN 1
data[9]:	glitch delay for FAN 2
data[10]:	glitch delay for FAN 3
data[11]:	glitch delay for FAN 4

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Read Fan Settings

Command Packet:

type:	37 (0x25)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	FBSCAB module index

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	12
data[0]:	1 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	power level for FAN 1 (0-100 valid)
data[3]:	power level for FAN 2 (0-100 valid)
data[4]:	power level for FAN 3 (0-100 valid)
data[5]:	power level for FAN 4 (0-100 valid)
data[6]:	fail-safe enabled for these fans bitmask
data[7]:	fan power update must happen within this many 1/8 second periods
data[8]:	glitch delay for FAN 1
data[9]:	glitch delay for FAN 2
data[10]:	glitch delay for FAN 3
data[11]:	glitch delay for FAN 4

Subcommand 2 - Read Fan Tachometers

This subcommand reads the last fan tachometer's information from the specified FBSCAB module. If this command is not re-executed within 60 seconds, fan speed readings will be disabled by the module to reduce fan noise until the next "Read Fan Tachometers" subcommand is issued.

See Appendix A for RPM calculation information and example source-code.
If fan tachometer readings are unstable or unreliable, see subcommand 1 for glitch delay adjustment.

Command Packet:

type:	37 (0x25)
data_length:	2
data[0]:	2 (subcommand number)
data[1]:	FBSCAB module index

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	14
data[0]:	2 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	fan 1 number of fan tach cycles
data[3]:	fan 1 of fan timer ticks (16bit, unsigned, low-byte)
data[4]:	fan 1 of fan timer ticks (16bit, unsigned, high-byte)
data[5]:	fan 2 number of fan tach cycles
data[6]:	fan 2 of fan timer ticks (16bit, unsigned, low-byte)
data[7]:	fan 2 of fan timer ticks (16bit, unsigned, high-byte)
data[8]:	fan 3 number of fan tach cycles
data[9]:	fan 3 of fan timer ticks (16bit, unsigned, low-byte)
data[10]:	fan 3 of fan timer ticks (16bit, unsigned, high-byte)
data[11]:	fan 4 number of fan tach cycles
data[12]:	fan 4 of fan timer ticks (16bit, unsigned, low-byte)
data[13]:	fan 4 of fan timer ticks (16bit, unsigned, high-byte)

Subcommand 3 - Read DOW Device Information

This command returns the ROM ID of the specified DOW (Dallas one wire) device attached to the specified FBSCAB module.

Command Packet:

type:	37 (0x25)
data_length:	2
data[0]:	3 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	DOW device index (0-15)

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	11
data[0]:	3 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	DOW device index
data[3-10]:	DOW ROM ID

Subcommand 4 - Read DOW Temperature Sensor Value

This command returns the temperature of a specified DOW device on a specified FBSCAB module. The specified DOW device must be of type 0x22 or 0x28 as read by command 37, subcommand 3.

Type 0x22 = Maxim DS18B20 sensor (as used by Crystalfontz WR-DOW-Y17)
Type 0x28 = Maxim DS1822 sensor

Command Packet:

type:	37 (0x25)
data_length:	2
data[0]:	4 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	DOW device index (0-15)

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	5
data[0]:	4 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	DOW device index
data[3]:	encoded temperature data (16bit, low-byte)
data[4]:	encoded temperature data (16bit, high-byte) bit0-10 = 11bit temperature value in degrees celcius * 16 bit11-13 = sign extension (2s complement) bit14-15 = DOW CRC status: 00 = CRC was checked and passed 01 = CRC was checked and failed 10 = no sensor detected in this index 11 = valid sensor, but no data read yet

Backwards Compatible: Yes (CFA835).

Subcommand 5 - FBSCAB GPIO Pin Levels

This command sets the GPIO pin levels of the FBSCAB. Do not confuse FBSCAB GPIOs with the GPIOs available on the module itself. This subcommand controls only the selected FBSCAB's GPIOs.
To use the module GPIOs see command 34.

The architecture of the FBSCAB allows flexibility in configuring the GPIO pins. They can be set as input or output. They can output constant high or low signals or a variable duty cycle 100Hz PWM signal. In output mode using the PWM (and a suitable current limiting resistor), a LED may be turned on or off and even dimmed under host software control. With suitable external circuitry, the GPIOs can also be used to drive external logic or power transistors. The FBSCAB continuously polls the GPIOs as inputs. The present level can be queried by the host software at a lower rate. The FBSCAB also keeps track of rising and falling edges since the last host query (subject to the resolution of the 50 Hz sampling), so the host is not forced to poll quickly in order to detect short events. The algorithm used by the FBSCABs to read the inputs is inherently debounced. The GPIOs also have "pull-up" and "pull-down" modes. These modes are useful the GPIO is an input connected to a switch, since no external pull-up or pull-down resistor is needed. For instance, the GPIO can be set to pull up. Then when a switch connected between the GPIO and ground is open, reading the GPIO will return a "1". When the switch is closed, the input will return a "0". Pull-up/pull-down resistance values are approximately 40kΩ. Typical GPIO current limits when sinking or sourcing all five GPIO pins simultaneously are 8 mA.

Set Pin Value & Configuration

Command Packet:

type: 37 (0x25)
data_length: 5
data[0]: 5 (subcommand number)
data[1]: FBSCAB module index
data[2]: index of GPIO to modify:
- 0 = J8 pin 7
- 1 = J8 pin 6
- 2 = J8 pin 5
- 3 = J8 pin 4
- 4 = J8 pin 2 (DOW I/O, always has 1K hardware pull-up)

data[3]: pin output value (behavior depends on drive mode) (0-100 valid)
- 0 = output set to low
- 1-99 = output PWM duty cycle percentage
- 100 = output set to high

data[4]: pin function and drive mode:
- bit0-2 = drive mode:
  - 000 = strong drive up, resistive pull down
  - 001 = strong drive up, strong drive down
  - 010 = hi-z (use for input)
  - 011 = resistive pull up, strong drive down
  - 100 = strong drive up, hi-z drive down
  - 101 = strong drive up, no drive down (hi-z)
  - 110 = reserved, do not use (returns error packet)
  - 111 = no drive up (hi-z), strong drive down
- bit3 = function:
  - 0 = pin unused
  - 1 = pin under user control
- bit4-7 = reserved (must be 0)

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	1
data[0]:	5 (subcommand number)

Read Pin Value & Configuration

Command Packet:

type:	37 (0x25)
data_length:	3
data[0]:	5 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	index of GPIO

Successful Return Packet:

type:	101 (0x25 | 0x40)
data_length:	6
data[0]:	5 (subcommand number)
data[1]:	FBSCAB module index
data[2]:	index of the GPIO read
data[3]:	pin state: bit0 = last sampled pin state bit1 = set if one or more falling edges have been detected on this pin since this command was last sent bit2 = set if one or more rising edges have been detected on this pin since this command was last sent bit3-7 = reserved
data[4]:	pin output value (0 to 100)
data[5]:	pin function and drive mode: bit0-2 = drive mode: 000 = strong drive up, resistive pull down 001 = strong drive up, strong drive down 010 = hi-z (use for input) 011 = resistive pull up, strong drive down 100 = strong drive up, hi-z drive down 101 = strong drive up, no drive down (hi-z) 110 = reserved, do not use (returns error packet) 111 = no drive up (hi-z), strong drive down bit3 = function: 0 = pin unused/default (see table above) 1 = pin under user control bit4-7 = reserved (must be 0)

Subcommand 6 - Reset and Search

This command sends a reset instruction to all attached FBSCAB modules. This reverts the FBSCAB modules back to their saved power-on state. After sending the reset instructions, the module re-searches for attached FBSCAB modules. For one attached FBSCAB, this command takes approximately 400mS to complete and return the response packet. For multiple FBSCABs, searching may take up to 2 additional seconds.

Command Packet:

type:	37 (0x25)
data_length:	1
data[0]:	6 (subcommand number)

Successful Return Packet:

type: 01 (0x25 | 0x40)
data_length: 1
data[0]: 6 (subcommand number)

Subcommand 7: Live Fan or Temperature Display

This command is not currently implemented. Please contact Crystalfontz for further information.

Subcommand 8: Automatic Fan Control

This command sets up automatic fan control based on a target temperature to operate without host intervention. With one or more attached FBSCABs, the module can be configured to automatically control fan power levels based upon the temperature of an attached DOW temperature sensor. The module will slow down or speed up the specified fan to attempt to maintain a set target temperature. Once configured, the module and FBSCAB will continue to automatically control fan speed without host/user intervention.

Fan control operation:

- If the specified temperature sensor's temperature is below the target value, the fan power will be gradually decreased at a rate determined by the responsiveness setting.
- If the specified temperature sensor's temperature is above the target value, the fan power will be gradually increased at a rate determined by the responsiveness setting.
- If the calculated fan power is below the specified minimum fan power value, the fan will either remain at that minimum value, or turn off depending on the "minimum fan power" option bits.
- If the calculated fan power is above the specified maximum fan power value, the fan will remain at the maximum fan power.
- If the specified temperature sensor does not exist (or there is a problem reading its value), the fan will be set to the specified maximum fan power value.

Each of the four fans attached to each attached FBSCAB module may be setup for automatic fan control. However, the temperature sensor used for a fan's power control must be attached to the same physical FBSCAB module as the fan. When automatic fan control is enabled for a fan, manual fan speed control (as set by subcommand 1) will be unavailable (command will succeed, but setting will be ignored). The power of the fan as set by automatic fan control may be read using subcommand 1 as normal.

Set Automatic Fan Control

Command Packet:

type: 37 (0x25)
data_length: 4 or 8
data[0]: 8 (subcommand number)
data[1]: FBSCAB module index
data[2]: controlled fan number (0-3 valid)
data[3]: option bits:
    bit0 = set to enable auto fan control
    bit1 = minimum fan power options:
        0 = if power is under minimum value, the minimum value is used
        1 = if power is under minimum value, fan power is turned off
    bit2 = reserved (always set to 0)
    bit3-7 = responsiveness value:
        0 = slow
        15 = fast
data[4]: monitored temp sensor DOW index (0-15 valid)
data[5]: target temperature + 128 (degrees celsius)
    -40 degrees = -40 + 128 = 88 (minimum valid value)

127 degrees = 127 + 128 = 255 (maximum valid value)
data[6]: minimum fan power value % (0-99 valid)
data[7]: maximum fan power value % (1-100 valid)
- must be higher than minimum value (data[6])
- also used for initial startup power value, and if specified temp sensor does not exist.

Successful Return Packet:

type: 101 (0x25 | 0x40)
data_length: 1
data[0]: 8 (subcommand number)

Read Automatic Fan Control

Command Packet:

type: 37 (0x25)
data_length: 3
data[0]: 8 (subcommand number)
data[1]: FBSCAB module index
data[2]: controlled fan number (0-3 valid)

Successful Return Packet:

type: 101 (0x25 | 0x40)
data_length: 8
data[0]: 8 (subcommand number)
data[1]: FBSCAB module index
data[2]: controlled fan number (0-3 valid)
data[3]: option bits (see above "Set Automatic Fan Control")
data[4]: monitored temp sensor DOW index (0-15 valid)
data[5]: target temperature + 128 (degrees celsius)
data[6]: minimum fan power value %
data[7]: maximum fan power value %

39 (0x27): Storage Command Group

This group provides commands to perform operations on the on-board flash storage, or with a microSD card inserted into the microSD slot on the rear of the module.

For all storage/file operation commands, a file name drive prefix of “a:\” refers to the on-board flash, and a prefix of “b:\” refers to the microSD card. If no drive prefix is given, on-board flash is the default storage device.

Subcommand 0 - Open or Close File

Opens the specified file for reading and/or writing. Only one file may be accessed at a time.
The subcommands 1 through 4 operate on the opened file.

Command Packet:

type: 39 (0x27)
data_length: 2 to 124
data[0]: 0 (subcommand number)
data[1]: file options:
0 = close currently opened file (file name does not need to be specified)
1 = open file for reading
2 = open file for reading and writing (truncates existing file)
3 = open file for reading and writing (appends to existing file)
data[2-123]: file name

File options 1 and 2 will set the file pointer position to the start of the file (position 0).
File option 3 will set the file pointer position to the end of the file.

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	5
data[0]:	0 (subcommand number)
data[1-4]:	file size in bytes (32bit, unsigned, little-endian)

Subcommand 1 - Position Seek

Seeks (sets the file pointer) to the location specified in the file opened by subcommand 0.

Command Packet:

type:	39 (0x27)
data_length:	5
data[0]:	1 (subcommand number)
data[1-4]:	file position (32bit, unsigned, little-endian)

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Subcommand 2 - Read File Data

Reads data from the file opened by subcommand 0. Data is read starting at the current file pointer location. The file pointer position is incremented by the amount of data read by this command. To read data from elsewhere in the file, use subcommand 1.

If the returned length of data read from the file is less than requested, the end of the file has been reached.

Command Packet:

type:	39 (0x27)
data_length:	5
data[0]:	2 (subcommand number)
data[1]:	number of bytes to read (1 to 123 valid)

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	2 to 124
data[0]:	2 (subcommand number)
data[1-123]:	data read from file

Subcommand 3 - Write File Data

Writes data to the file opened by subcommand 0. Data is written starting at the current file pointer location.

Note:
Writing data to the on-board flash is quite slow in comparison to a typical microSD card.

Command Packet:

type:	39 (0x27)
data_length:	2 to 124

data[0]:	3 (subcommand number)
data[1-123]:	data to write to the file

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	1
data[0]:	3 (subcommand number)

Subcommand 4 - Delete A File

Deletes the specified file from the storage filesystem. Attempting to delete a currently open file will result in an error.

Command Packet:

type:	39 (0x27)
data_length:	2 to 124
data[0]:	4 (subcommand number)
data[1-123]:	name of the file to delete

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	1
data[0]:	4 (subcommand number)

Subcommand 5 - File Copy

This command copies the specified file name located on the on-board flash or microSD card filesystem to the specified destination filename on the on-board flash or microSD card.
It can be used to copy files to the same filesystem, or from the microSD card to the on-board flash filesystem or vice versa.

This command is unique in that it must be sent twice to complete the file copy operation.
The first command packet sent specifies the source file name. The second packet specifies the destination file name. After the destination file name has been specified, if both file names are valid, the file will then be copied.

This command can take quite some time to return depending on the size of the file being copied and the destination filesystem type (writing to the on-board flash is quite slow). For example, a file copied from microSD to On-board flash of 500kb in size may take 30 seconds.
While the file copying is in progress, the module will not process any other incoming packets from the host. The module will send the return packet to the host when the file copy process has been completed.

Command Packet:

type:	39 (0x27)
data_length:	2 to 124
data[0]:	5 (subcommand number)
data[1]:	options: 0 = set source file name 1 = set destination file name
data[2-123]:	source or destination file name

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	1
data[0]:	5 (subcommand number)

Subcommand 6 - File System Erase/Format

This command formats the specified file system, erasing all files.

This command can take quite some time to return. While the file system format is in progress, the module will not process any other incoming packets from the host. The module will send the return packet to the host when the format process has been completed.

Warning: All opened files (video, ttf fonts, cmd39) must be closed before sending this command. Not doing so will produce unpredicable results. It is advised that the module is reset (command 5), immediately after issuing this command to avoid problems.

Command Packet:

type:	39 (0x27)
data_length:	2
data[0]:	6 (subcommand number)
data[1]:	file system to format: 0 = on-board flash (littlefs) 1 = microsd card (fat32)

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	1
data[0]:	6 (subcommand number)

Subcommand 7 - Storage Capacity/Available

Use this command to query the total capacity and available space remaining on the specified file system or memory device.

Command Packet:

type:	39 (0x27)
data_length:	2
data[0]:	7 (subcommand number)
data[1]:	storage / memory selection: 0 = on-board flash 1 = microsd card 2 = graphics memory (ram) 3 = os/program memory (ram)

Successful Return Packet:

type:	103 (0x27 | 0x40)
data_length:	10
data[0]:	7 (subcommand number)
data[1]:	storage selection (as above)
data[2-5]:	total capacity in bytes (32bit, unsigned, little-endian)
data[6-9]:	available space in bytes (32bit, unsigned, little-endian)

40 (0x28) Graphics Core Command Group

See the Graphics section for information on the modules graphics system.

Subcommand 0 - Set Graphics Core Options

Controls the display update mode of the module.
If the mode is automatic, the display will be updated any time a graphical object, the legacy layer, or module status layer

is updated.
In manual mode, the display will only be updated when the “buffer flush” (subcommand 1) is sent to the module by the host.

Command Packet:

type:	40 (0x28)
data_length:	2
data[0]:	0 (subcommand number)
data[1]:	option flags: bit0 = display update mode (0 = auto, 1 = manual) bit1 = (reserved, for CFA835 backwards compatibility)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Manual Display Update

This command manually triggers updating the modules display.
This command has no effect unless subcommand 0 display update mode option is set to manual.

Command Packet:

type:	40 (0x28)
data_length:	1
data[0]:	1 (subcommand number)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Subcommand 2 - Set Background Color

Sets the color of the background of the display.
All other display objects and layers are placed on top of this color. The change is immediate.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	2 (subcommand number)
data[1-2]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	2 (subcommand number)

Subcommand 2 - Get Background Color

Returns the color setting of the background of the display.

Command Packet:

type:	40 (0x28)
data_length:	1
data[0]:	2 (subcommand number)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	3
data[0]:	2 (subcommand number)
data[1-2]:	RGB565 color value (16bit, unsigned, little-endian)

Subcommand 10 - Remove Graphics Object

Removes/deletes a previously create graphics object.
After removal, the graphics object ID number becomes unused.

See command 6 to remove all objects, and clear the display.

Command Packet:

type:	40 (0x28)
data_length:	2
data[0]:	10 (subcommand number)
data[1]:	graphics object ID number (1-255 valid)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	10 (subcommand number)

Subcommand 11 - Move Graphics Object

Move the specified graphics object to the specified pixel location.
If the graphics object is moved partially outside of the the visibile region of the display (480x128), it will be clipped accordingly.
If the object is moved completely outside the visible region of the display, it will not be rendered (this is one method of temporarily hiding a graphics object without removing it).

Command Packet:

type:	40 (0x28)
data_length:	6
data[0]:	11 (subcommand number)
data[1]:	graphics object ID number (1-255 valid)
data[2-3]:	new X position (16bit, signed, little-endian)
data[4-5]:	new Y position (16bit, signed, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	11 (subcommand number)

Subcommand 12 - Set Graphics Object Touch Reporting

If enabled by this command, a graphics object when touched (requires the optional touch-screen display), can send a report packet to the host detailing touch information.
The reported X,Y pixel location is relative to the location of the graphics object.

Note:
Report packets must be enabled on the communications interface for these report packets to be recieved by the host. See command 33 for details.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	12 (subcommand number)
data[1]:	graphics object ID number (1-255 valid)
data[2]:	touch reporting option: 0 = off 1 = report inital touch and release (including positions) 2 = report inital touch, position updates and release

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	12 (subcommand number)

Report Packet:

type:	132 (0x04 | 0x80)
data_length:	8
data[0]:	touched graphics object ID number
data[1]:	count of points (always 1)
data[2]:	track id of touch point (always 0)
data[3]:	status flag: bit0 = released bit1 = pressed bit2 = held/dragged
data[4-5]:	x position (16bit, unsigned, little-endian)
data[6-7]:	y position (16bit, unsigned, little-endian)

Subcommand 13 - Set Graphics Object Z-Index

Set the Z-Index of the specified graphics object.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	13 (subcommand number)
data[1]:	graphics object ID number (1-255 valid)
data[2]:	new z-index value

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	13 (subcommand number)

Subcommand 15 - Style: True-Type Font Slot Selection

Set the true-type font slot used by the default style, or the specified graphics object.
See command 43 for true-type font use and configuration details.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	15 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2]:	true-type font slot number

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	15 (subcommand number)

Subcommand 16 - Style: True-Type Font Size

Set the true-type font size used by the default style, or the specified graphics object.
See command 43 for true-type font use and configuration details.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	16 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2]:	true-type font size

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	16 (subcommand number)

Subcommand 17 - Style: Corner Radius

Sets the outside border corner radius used by the default style, or the specified graphics object.

Note:
This setting is not error-checked during rendering. Using overly large values for the size of the corner radius may produce visual rendering errors.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	17 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2]:	corner radius (in pixels)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	17 (subcommand number)

Subcommand 18 - Style: Object Opacity

Sets the default style or specified graphics objects opacity (transparency) value.
An object with a value of 0 (zero) is completley transparent, and will not be rendered.

Command Packet:

type:	40 (0x28)
data_length:	3
data[0]:	18 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2]:	new opacity value (0-255 valid)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	18 (subcommand number)

Subcommand 19 - Style: Fill-A Color

Sets the color of the default style or graphics objects Fill-A property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	19 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	19 (subcommand number)

Subcommand 20 - Style: Border-A Color

Sets the color of the default style or graphics objects Border-A property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	20 (subcommand number)
data[1]:	object number: 0 = default style

1-255 = graphics object ID
data[2-3]: RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type: 104 (0x28 | 0x40)
data_length: 1
data[0]: 20 (subcommand number)

Subcommand 21 - Style: Control-A Color

Sets the color of the default style or graphics objects Control-A property.
See the graphics object type creation command for details.

Command Packet:

type: 40 (0x28)
data_length: 4
data[0]: 21 (subcommand number)
data[1]: object number:
0 = default style
1-255 = graphics object ID
data[2-3]: RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type: 104 (0x28 | 0x40)
data_length: 1
data[0]: 21 (subcommand number)

Subcommand 22 - Style: Text-A Color

Sets the color of the default style or graphics objects Text-A property.
See the graphics object type creation command for details.

Command Packet:

type: 40 (0x28)
data_length: 4
data[0]: 22 (subcommand number)
data[1]: object number:
0 = default style
1-255 = graphics object ID
data[2-3]: RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type: 104 (0x28 | 0x40)
data_length: 1
data[0]: 22 (subcommand number)

Subcommand 23 - Style: Fill-B Color

Sets the color of the default style or graphics objects Fill-B property.
See the graphics object type creation command for details.

Command Packet:

type: 40 (0x28)
data_length: 4
data[0]: 23 (subcommand number)

data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	23 (subcommand number)

Subcommand 24 - Style: Border-B Color

Sets the color of the default style or graphics objects Border-B property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	24 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	24 (subcommand number)

Subcommand 25 - Style: Control-B Color

Sets the color of the default style or graphics objects Control-B property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	25 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	25 (subcommand number)

Subcommand 26 - Style: Text-B Color

Sets the color of the default style or graphics objects Text-B property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	26 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	26 (subcommand number)

Subcommand 27 - Style: Fill-C Color

Sets the color of the default style or graphics objects Fill-C property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	27 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	27 (subcommand number)

Subcommand 28 - Style: Border-C Color

Sets the color of the default style or graphics objects Border-C property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	28 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	28 (subcommand number)

Subcommand 29 - Style: Control-C Color

Sets the color of the default style or graphics objects Control-C property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	29 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	29 (subcommand number)

Subcommand 30 - Style: Text-C Color

Sets the color of the default style or graphics objects Text-C property.
See the graphics object type creation command for details.

Command Packet:

type:	40 (0x28)
data_length:	4
data[0]:	30 (subcommand number)
data[1]:	object number: 0 = default style 1-255 = graphics object ID
data[2-3]:	RGB565 color value (16bit, unsigned, little-endian)

Successful Return Packet:

type:	104 (0x28 | 0x40)
data_length:	1
data[0]:	26 (subcommand number)

41 (0x29) Video Object Command Group

The module has the ability to multiple play full or partial screen, color videos at up to 30fps, while also displaying other graphics objects.
These videos can be used for backgrounds (controls can be placed on top), animated controls, instructions, or any other number of uses.

Video files use a standard AVI file format, which are stored on either the on-board flash, or a microSD card.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Notes: - The video file must be of motion-jpeg (MJPEG) AVI type, using YUV420p encoding, have no audio track, and must be of smaller pixel width/height than the display (480x128).
- The module attempts to play the video at the frame-rate encoded in the AVI file, however the frame-rate may decrease if the module becomes busy performing other tasks or has many other graphical objects to display. - If the end frame number is not 0, it must be higher than the starting frame number. - If the starting frame is not 0, there may be a short delay when starting playback, or looping back to the starting frame.
- On loading of saved module state (power-on, or reset command), videos will load from the starting frame number.

Subcommand 0 - Video File Load/Play

This command creates a new video playback graphics object using the specified motion-jpeg (AVI) video file.

Style options used are:

- Video opacity = Style: Object Opacity

Command Packet:

type:	41 (0x29)
data_length:	15 to 124
data[0]:	0 (subcommand number)
data[1]:	new graphics object ID number (1-255 valid)
data[2]:	option flags: bit0 = starting play state (1=play, 0=paused)
data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y Position (16bit, signed, little-endian)
data[7]:	z-index
data[8]:	touch reporting option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[9-10]:	video starting frame number (16bit, unsigned, little-endian)
data[11-12]:	video ending frame number (16bit, unsigned, little-endian) 0 = play to the end of the file
data[13]:	play video x times in a loop (0 = continuous)
data[14+]:	video file name

Successful Return Packet:

type:	105 (0x29 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Video Play Control

This command changes the playback options of a previously created video graphics object.

Playing from the start, or un-pausing playback resets the current loop counter.
Setting a new loop count resets the current loop counter.
If the starting frame is not 0, there may be a short delay when starting playback, or looping back to the starting frame.

Command Packet:

type:	41 (0x29)
data_length:	3 or 7 or 8
data[0]:	1 (subcommand number)
data[1]:	existing video graphics object ID number (1-255 valid)
data[2]:	video control option: 0 = no change 1 = play from the starting frame number 2 = pause playback (toggles)
data[3-4]:	video starting frame number (16bit, unsigned, little-endian)
data[5-6]:	video ending frame number (16bit, unsigned, little-endian) 0 = play to the end of the file
data[7]:	play video x times in a loop (0 = continuous)

Successful Return Packet:

type:	105 (0x29 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

42 (0x2A) Image Object Command Group

The module can display multiple images, from a variety of file formats which are located on either the on-board flash, or microSD card.

The image object also has the ability to act as a touch button, with a different image being used for each button state (up, down, disabled). The state of the button can also be changed by using subcommand 4.

When loaded, image data is decoded and saved in the graphics ram (16MB capacity) until the object is removed (see command 6 or command 40, subcommand 10). The amount of memory used by the image depends on the image format.
For example:

- 100x100 pixel JPG image will be decoded as RGB565 (16bits/pixel) and will use 100x100x2 = 20,000 bytes of graphics memory.

- 200x128 pixel PNG image with an alpha transparency layer will be decoded as ARGB8888 (32bits/pixel) and will use 200x128x4 = 102,000 bytes of graphics memory.

See command 39, subcommand 7, or the debug log using command 62 for monitoring graphics memory use.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Notes:

- Supports JPG,PNG,BMP formats (max 16bits per channel).
- Hardware decoding for JPG files (fast).
- Alpha layers in image file are used if present.
- Files are kept loaded/decoded into RAM (RAM space is limited) until removed.
- Images are displayed pixel-for-pixel on display. No image resizing or other manipulation is possible.
- Maximum image size X or Y is 2048 pixels.

Subcommand 0 - Image File Load / Change Option Flags

Loads an image file, and creates an image graphics object for static image or button (up state) use.

Style options used are:

- Image opacity = Style: Object Opacity

Image File Load

Command Packet:

type:	42 (0x2A)
data_length:	11 to 124
data[0]:	0 (subcommand number)
data[1]:	new graphics object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = is disabled (uses disabled state image if loaded) bit1 = is a button (down and disabled state images must be loaded) bit2 = is a toggle button (not momentary) bit3 = create in the down position (toggle type only) bit4 = enable button state change report packets
data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y Position (16bit, signed, little-endian)

data[7]: z-index
data[8]: touch reporting option:
0 = off
1 = report single touch & release (including position)
2 = report single touch, position updates and release
data[9+]: image file name

Successful Return Packet:

type: 106 (0x2A | 0x40)
data_length: 1
data[0]: 0 (subcommand number)

Change Image Option Flags

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type: 42 (0x2A)
data_length: 3
data[0]: 0 (subcommand number)
data[1]: existing image graphics object ID number
data[2]: option flags:
bit0 = is disabled (uses disabled state image if loaded)
bit1 = is a button (down and disabled state images must be loaded)
bit2 = is a toggle button (not momentary)
bit3 = create in the down position (toggle type only)
bit4 = enable button state change report packets

Successful Return Packet:

type: 106 (0x2A | 0x40)
data_length: 1
data[0]: 0 (subcommand number)

Subcommand 1 - Button Image File Load (Down State)

This command loads an image file for the button down state.

Command Packet:

type: 42 (0x2A)
data_length: 3 to 124
data[0]: 1 (subcommand number)
data[1]: existing image graphics object ID number
data[3+]: image file name

Successful Return Packet:

type: 106 (0x2A | 0x40)
data_length: 1
data[0]: 1 (subcommand number)

Subcommand 2 - Button Image File Load (Disabled State)

This command loads an image file for the button disabled state.

Command Packet:

type:	42 (0x2A)
data_length:	3 to 124
data[0]:	2 (subcommand number)
data[1]:	existing image graphics object ID number
data[3+]:	image file name

Successful Return Packet:

type:	106 (0x2A | 0x40)
data_length:	1
data[0]:	2 (subcommand number)

Subcommand 3 - Image Load From Host

This command allows the host to send image data to the module creating an image object. The data is sent to the host using a “data streaming” mode that is unique to this command.

After this command packet has been sent to the module, image data (not in normal packet format) is sent to the module on the same communications interface. When the module has recieved the image data (of size specified in the command), it will send the acknowledge packet to the host, and create the image graphics object.
If the image data of correct size is not transfered before the timeout occours, an error packet will be sent to the host, then normal packet communications on the interface will resume. As the image data is not sent inside packets and has no error checking, any data errors on the communications interface will result in no image, or an incorrect image being displayed on the module.

Monochrome and greyscale data formats are rendered using Style Text-A color.

On saving module state (command 4), image data will be saved to files on the on-board flash. If not enough space is available on the on-board flash, the save state command (command 4) will fail.

This command does not support creation of a stateful button type image object.

The image data transmission timeout varies depending on the communications interface being used.
The timeout values are 3 seconds for the USB and SPI interfaces, and 5 seconds for the serial and I2C interfaces.

Data format information:

format index	format name	format data size	internally converted to	used graphics memory
0	raw monochrome	1bit/pixel	8bits/pixel alpha	(width * height) bytes
1	raw greyscale	8bits/pixel	no conversion	(width * height) bytes
2	raw RGB565	16bits/pixel	no conversion	(width * height * 2) bytes
3	raw ARGB1555	16bits/pixel	no conversion	(width * height * 2) bytes
4	raw ARGB8888	32bits/pixel	no conversion	(width * height * 4) bytes
5	JPG	encoded	16bits/pixel RGB565	(width * height * 2) bytes
5	other	encoded (no alpha channel)	16bits/pixel RGB565	(width * height * 2) bytes
5	other	encoded (has alpha channel)	32bits/pixel ARGB8888	(width * height * 4) bytes

Style options used are:

- Image opacity = Style: Object Opacity

Command Packet:

type:	42 (0x2A)
data_length:	18
data[0]:	3 (subcommand number)

data[1]:	new graphics object ID number (1-255 valid)
data[2]:	option flags (reserved)
data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y position (16bit, signed, little-endian)
data[7]:	z-index
data[8]:	touch reporting option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[9]:	data format: 0 = raw monochrome (1-bit per pixel) 1 = raw greyscale (1-byte/8-bits per pixel) 2 = raw RGB565 (16-bits per pixel) 3 = raw ARGB1555 (16-bits per pixel, 1-bit alpha) 4 = raw ARGB8888 (32-bits per pixel) 5 = jpg/png/bmp file data (whole file, including file header)
data[10-11]:	image width (16bit, unsigned, little-endian)
data[12-13]:	image height (16bit, unsigned, little-endian)
data[14-17]:	image data size (32bit, unsigned, little-endian)

Successful Return Packet:

type:	106 (0x2A | 0x40)
data_length:	1
data[0]:	3 (subcommand number)

Subcommand 4 - Set / Read Image Toggle Button State

Set Image Toggle Button State

Note:
Using this command on an image that is not a toggle button type, will succeed but have no effect.

Command Packet:

type:	42 (0x2A)
data_length:	3
data[0]:	4 (subcommand number)
data[1]:	existing image graphics object ID number
data[2]:	new state: 0 = up 1 = down

Successful Return Packet:

type:	106 (0x2A | 0x40)
data_length:	1
data[0]:	4 (subcommand number)

Read/Poll Image Button State

Note:
Reading the state of a image object that is not a button type, will succeed but always return a state value of 0.

Command Packet:

type:	42 (0x2A)
data_length:	2
data[0]:	4 (subcommand number)
data[1]:	existing image graphics object ID number

Successful Return Packet:

type:	106 (0x2A | 0x40)
data_length:	2
data[0]:	4 (subcommand number)
data[1]:	graphics object ID number
data[2]:	state flags: bit0 = current state (0=up, 1=down) bit1 = has been pressed since last poll bit2 = has been released since last poll

43 (0x2B) True Type Fonts Command Group

The module supports direct use of standard true-type font (TTF) files for text rendering on the display. Up to 15 font files may loaded into “font slots” from the on-board flash or microSD storage at a time.

A built-in font is also always available for use, in font slot 0 (zero). It is a trimmed down version of the JetBrains Mono NL TTF font.

Font files are loaded completley into graphics ram (16MB capacity), so the use of compact font files is recomended. Font files have a 2.5MB size limit.
See command 39, subcommand 7, or the debug log using command 62 for monitoring graphics memory use.
True-type font files often contain many extended character glyphs which can use a lot of extra storage/memory space. If these extra glyphs are not needed, consider using a true-type font editor (like FontForge) to remove them.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - Load True Type Font

This command loads the specified true-type font (TTF) file from the on-board flash or a microSD card into a “font slot” which is then used for displaying text on the modules display by subcommand 2, or some of the other graphics object related commands.

Command Packet:

type:	43 (0x2B)
data_length:	4 to 124
data[0]:	0 (subcommand number)
data[1]:	font slot number (1 to 15 valid, 0 is reserved)
data[2]:	option flags (reserved)
data[3+]:	file name (on-board flash or microsd card)

Successful Return Packet:

type:	107 (0x2B | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Unload True Type Font

This command unloads a previously loaded true-type font.

Command Packet:

type:	43 (0x2B)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	font slot number (1 to 15 valid, 0 is reserved)

Successful Return Packet:

type:	107 (0x2B | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Subcommand 2 - New Text Object (Display Text)

This command creates a text graphics object.

The font slot, font size, and color used are taken from the current default style options, see command 40.

This command will return an error packet if the Object ID number is already used, unless option bit0 is set.

If neither the center or right justified option flags are set, the text will be left justified.

Style options used are:

- Text font = Style: True-Type Font Slot
- Text size = Style: True-Type Font Size
- Text color = Style: Text-A Color
- Text opacity = Style: Object Opacity

Command Packet:

type:	43 (0x2B)
data_length:	10 to 124
data[0]:	2 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = dont create object (return x/y pixel size only) bit1 = 0=ASCII/UTF8, 1=UTF16 bit2 = reserved bit3 = center justified bit4 = right justified
data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y Position (16bit, signed, little-endian)
data[7]:	z-index
data[8]:	touch reporting: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[9+]:	text to display (utf8 or utf16)

Successful Return Packet:

type:	107 (0x2B | 0x40)
data_length:	1
data[0]:	2 (subcommand number)

Subcommand 3 - Modify Text Object

This command modifies the options and/or text of an existing text graphics object.
See command 40 for changing other properties of this graphics object.

Command Packet:

type:	43 (0x2B)
data_length:	4 to 124
data[0]:	3 (subcommand number)

data[1]:	existing text graphics object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = dont create object (return x/y pixel size only) bit1 = 0=ASCII/UTF8, 1=UTF16 bit2 = reserved bit3 = center justified bit4 = right justified
data[3+]:	new text to display (utf8 or utf16)

Successful Return Packet:

type:	107 (0x2B | 0x40)
data_length:	1
data[0]:	4 (subcommand number)

44 (0x2C) Sketch Surface Command Group

The sketch surface type graphics object provides the ability to draw simple graphics on the display.

Create the sketch object, then draw lines, rectangles and circles on it.
This graphics object may also be useful for creating solid backgrounds for other objects, like text.

When created, each sketch surface graphic object is stored in the graphics ram (16MB capacity) until it is removed. The sketch uses 16bits/pixel, so for example, a 300x100 pixel sketch surface will use 300x100x2 = 60,000 bytes of graphics ram.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Sketch Surface

Create a new sketch surface graphics object.
The sketch surface can either be created with a filled background color, or completely transparent.

Style options used are:

- Sketch surface opacity = Style: Object Opacity
- Sketch surface fill color = Style: Fill-A Color

Command Packet:

type:	44 (0x2C)
data_length:	13
data[0]:	0 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = fill with color (style fill-a color)
data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y position (16bit, signed, little-endian)
data[7-8]:	width (16bit, unsigned, little-endian)
data[9-10]:	height (16bit, unsigned, little-endian)
data[11]:	z-index
data[12]:	touch reporting option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release

Successful Return Packet:

type:	108 (0x2C | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Draw Line on Sketch Surface

Draws a line on the sketch surface graphics object.

Command Packet:

type:	44 (0x2C)
data_length:	13
data[0]:	1 (subcommand number)
data[1]:	existing sketch surface object ID number (1 to 255 valid)
data[2]:	option flags (reserved)
data[3-4]:	x start position (16bit, unsigned, little-endian)

data[5-6]: y start position (16bit, unsigned, little-endian)
data[7-8]: x end position (16bit, unsigned, little-endian)
data[9-10]: y end position (16bit, unsigned, little-endian)
data[11-12]: RGB565 color (16bit, unsigned, little-endian)

Successful Return Packet:

type: 108 (0x2C | 0x40)
data_length: 1
data[0]: 1 (subcommand number)

Subcommand 2 - Draw Rectangle on Sketch Surface

Draws a rectangle on the sketch surface graphics object.

Command Packet:

type: 44 (0x2C)
data_length: 16
data[0]: 2 (subcommand number)
data[1]: existing sketch surface object ID number (1 to 255 valid)
data[2]: option flags:
bit0 = fill with color
bit1 = draw border
data[3-4]: x start position (top-left corner) (16bit, unsigned, little-endian)
data[5-6]: y start position (top-left corner) (16bit, unsigned, little-endian)
data[7-8]: width (16bit, unsigned, little-endian)
data[9-10]: height (16bit, unsigned, little-endian)
data[11]: corner radius
data[12-13]: fill RGB565 color (16bit, unsigned, little-endian)
data[14-15]: border RGB565 Color (16bit, unsigned, little-endian)

Successful Return Packet:

type: 108 (0x2C | 0x40)
data_length: 1
data[0]: 2 (subcommand number)

Subcommand 3 - Draw Circle on Sketch Surface

Draws a circle on the sketch surface graphics object.

Command Packet:

type: 44 (0x2C)
data_length: 13
data[0]: 3 (subcommand number)
data[1]: existing sketch surface object ID number (1 to 255 valid)
data[2]: option flags:
bit0 = fill with color
bit1 = draw border
data[3-4]: center x position (16bit, unsigned, little-endian)
data[5-6]: center y position (16bit, unsigned, little-endian)
data[7-8]: radius (16bit, unsigned, little-endian)
data[9-10]: fill RGB565 color (16bit, unsigned, little-endian)
data[11-12]: border RGB565 Color (16bit, unsigned, little-endian)

Successful Return Packet:

type:	108 (0x2C | 0x40)
data_length:	1
data[0]:	3 (subcommand number)

Subcommand 4 - Draw a Pixel on Sketch Surface

Draws a pixel on the sketch surface graphics object.

Command Packet:

type:	44 (0x2C)
data_length:	8
data[0]:	4 (subcommand number)
data[1]:	existing sketch surface object ID number (1 to 255 valid)
data[2-3]:	x position (16bit, unsigned, little-endian)
data[4-5]:	y position (16bit, unsigned, little-endian)
data[6-7]:	RGB565 color (16bit, unsigned, little-endian)

Successful Return Packet:

type:	108 (0x2C | 0x40)
data_length:	1
data[0]:	4 (subcommand number)

47 (0x2F) Button Command Group

This command group allows the user to create simple button touch input graphical objects on the modules display. The buttons are created using paramaters passed in the “new button object” subcommand, and style settings set by command 40.

The button supports up, down and disable states, and can operate as a momentary push-button or an on/off toggle button. Each of the states has configurable colors, and text labels which can be changed at any time.

See Image Command Group, command 42, for custom image based buttons.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Button Object / Change Option Flags

New Button Object

This command creates a new touch-screen operated push or toggle button graphical object on the display.

It’s various style options (colors, etc) can be set using command 40 (see the list below).
All style preference are adopted by the new object at time of creation. After creation, style options are unique to the object.

Option flags, bit4 enables state report packets to be sent to the host on button state change. This differs from the “touch report options” which report object touch locations only.

Special drawn symbols are available by using the following key words as the label text:

- Square symbol = “SQUARE”
- Circle symbol = “CIRCLE”
- Up triangle symbol = “UPTRI”
- Down triangle symbol = “DOWNTRI”
- Left triangle symbol = “LEFTTRI”
- Right triangle symbol = “RIGHTTRI”

Style options used are:

- Button opacity = Style: Object Opacity
- Button label text style = Style: True-Type Font Slot Selection
- Button label text size = Style: True-Type Font Size
- Button corner radius = Style: Corner Radius
- Up-state background fill color = Style: Fill-A Color
- Up-state border color = Style: Border-A Color
- Up-state text color = Style: Text-A Color
- Down-state background fill color = Style: Fill-B Color
- Down-state border color = Style: Border-B Volor
- Down-state text color = Style: Text-B Color
- Disabled-state background fill color = Style: Fill-C Color
- Disabled-state border color = Style: Border-C Color
- Disabled-state text color = Style: Text-C Color

Command Packet:

type: 47 (0x2F)
data_length: 14 to 124
data[0]: 0 (subcommand number)
data[1]: new object ID number (1 to 255 valid)
data[2]: option flags:
- bit0 = create in a disabled state
- bit1 = is a toggle button (not momentary)
- bit2 = create in the down position (toggle type only)
- bit3 = UTF16 text labels
- bit4 = enable state change report packets (see report packet below)

data[3-4]: x position (16bit, signed, little-endian)
data[5-6]: y position (16bit, signed, little-endian)
data[7-8]: width (16bit, unsigned, little-endian)
data[9-10]: height (16bit, unsigned, little-endian)
data[11]: z-index
data[12]: touch report option:
- 0 = off
- 1 = report single touch & release (including position)
- 2 = report single touch, position updates and release

data[13+]: text label (for up state)

Successful Return Packet:

type: 111 (0x2F | 0x40)
data_length: 1
data[0]: 0 (subcommand number)

Report Packet:

type: 134 (0x06 | 0x80)
data_length: 2
data[0]: button object ID number
data[1]: event type:
- 0 = button state is up
- 1 = button state is down
- 2 = toggle button state is down

Change Button Option Flags

Use this command to update the option flags after a button has been created.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type: 47 (0x2F)
data_length: 3
data[0]: 0 (subcommand number)
data[1]: existing button object ID number (1 to 255 valid)
data[2]: option flags:
- bit0 = set to disabled state
- bit1 = is a toggle button (not momentary)
- bit2 = create in the down position (toggle type only)
- bit3 = UTF16 text labels
- bit4 = enable state change report packets

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Set Up State Text Label

Set a new up state text label. If the UTF16 option bit was set during button object creation, it also applies here. The special drawn symbols (as listed above) can be used for this command.

Command Packet:

type:	47 (0x2F)
data_length:	3 to 124
data[0]:	1 (subcommand number)
data[1]:	existing button object ID number (1 to 255 valid)
data[2+]:	new text label (for up state)

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Subcommand 2 - Set Down State Text Label

Set a new down state text label. If the UTF16 option bit was set during button object creation, it also applies here. The special drawn symbols (as listed above) can be used for this command.

Command Packet:

type:	47 (0x2F)
data_length:	3 to 124
data[0]:	2 (subcommand number)
data[1]:	existing button object ID number (1 to 255 valid)
data[2+]:	new text label (for down state)

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	1
data[0]:	2 (subcommand number)

Subcommand 3 - Set Disabled State Text Label

Set a new disabled state text label. If the UTF16 option bit was set during button object creation, it also applies here. The special drawn symbols (as listed above) can be used for this command.

Command Packet:

type:	47 (0x2F)
data_length:	3 to 124
data[0]:	3 (subcommand number)
data[1]:	existing button object ID number (1 to 255 valid)
data[2+]:	new text label (for disabled state)

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	1
data[0]:	3 (subcommand number)

Subcommand 4 - Set / Read Toggle Button State

Set Toggle Button State

If the button object was created with the toggle option bit set, this command can alter the toggled state. Setting the up or down state on a non-toggle button, will succeed but have no effect.

Command Packet:

type:	47 (0x2F)
data_length:	3
data[0]:	4 (subcommand number)
data[1]:	existing button object ID number (1 to 255 valid)
data[2]:	new toggle button state: 0 = up state 1 = down state

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	1
data[0]:	4 (subcommand number)

Read/Poll Button State

This command allows the host to read the current state of the button.

Command Packet:

type:	47 (0x2F)
data_length:	2
data[0]:	4 (subcommand number)
data[1]:	existing button object ID number (1 to 255 valid)

Successful Return Packet:

type:	111 (0x2F | 0x40)
data_length:	3
data[0]:	4 (subcommand number)
data[1]:	button object ID number (1 to 255 valid)
data[2]:	state flags: bit0 = current state (0=up, 1=down) bit1 = has been pressed since last poll bit2 = has been released since last poll

48 (0x30) Slider Control Command Group

This command group allows the user to create touch-screen operated slider control graphical objects on the modules display. The slider controls are created using paramaters passed in the “new slider control object” subcommand, and style settings set by command 40.

The slider control supports enabled and disabled states, and has configurable start and end values. Each of the states has configurable colors which can be changed at any time.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Slider Control Object / Change Option Flags

New Slider Control Object

This command creates a new touch-screen operated slider control graphical object on the display.

It’s various style options (colors, etc) can be set using command 40 (see the list below).
All style preference are adopted by the new object at time of creation. After creation, style options are unique to the object.

Option flags, bit5 enables position report packets to be sent to the host on slider position change. This differs from the “touch report options” which report object touch locations only.

Style options used are:

- Slider opacity = Style: Object Opacity
- Slider label text style = Style: True-Type Font Slot Selection
- Slider label text size = Style: True-Type Font Size
- Slider corner radius = Style: Corner Radius
- Enabled-state Background fill color = Style: Fill-A Color
- Enabled-state Border color = Style: Border-A Color
- Enabled-state Text label color = Style: Text-A Color
- Enabled-state Text value color = Style: Text-B Color
- Enabled-state Slider bar color = Style: Control-A Color
- Enabled-state Slider position color = Style: Control-B Color
- Disabled-state Background fill color = Style: Fill-C Color
- Disabled-state Border color = Style: Border-C Color
- Disabled-state Text label color = Style: Text-C Color
- Disabled-state Text value color = Style: Text-C Color
- Disabled-state Slider bar color = Style: Control-C Color
- Disabled-state Slider position color = Style: Control-C Color

Command Packet:

type:	48 (0x30)
data_length:	20 to 124
data[0]:	0 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = show current value on the left bit2 = show current value on the right bit3 = no background fill

bit4 = UTF-16 text labels
bit5 = enable position change report packets (see report packet below)

data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y position (16bit, signed, little-endian)
data[7-8]:	width (16bit, unsigned, little-endian)
data[9-10]:	height (16bit, unsigned, little-endian)
data[11]:	z-index
data[12]:	touch report option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[13-14]:	minimum value (16bit, signed, little-endian)
data[15-16]:	maximum value (16bit, signed, little-endian)
data[17-18]:	starting value (16bit, signed, little-endian)
data[19+]:	text label

Successful Return Packet:

type:	112 (0x30 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Report Packet:

type:	135 (0x07 | 0x80)
data_length:	2
data[0]:	slider object ID number
data[1-2]:	updated value (16bit, signed, little-endian)

Change Slider Control Option Flags

Use this command to update the option flags after a slider control has been created.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type:	48 (0x30)
data_length:	3
data[0]:	0 (subcommand number)
data[1]:	existing slider control object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = show current value on the left bit2 = show current value on the right bit3 = no background fill bit4 = UTF-16 text labels bit5 = enable position change report packets (see report packet below)

Successful Return Packet:

type:	112 (0x30 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Set / Read Slider Control Value

Set Slider Control Value

Update the slider control value.

Command Packet:

type:	48 (0x30)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	existing slider control object ID number (1 to 255 valid)
data[2-3]:	new value (16bit, signed, little-endian)

Successful Return Packet:

type:	112 (0x30 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Read Slider Control Value

Read the slider control value.

Command Packet:

type:	48 (0x30)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	existing slider control object ID number (1 to 255 valid)

Successful Return Packet:

type:	112 (0x30 | 0x40)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	slider control object ID number
data[2-3]:	current value (16bit, signed, little-endian)

49 (0x31) Number-Edit Control Command Group

This command group allows the user to create simple number editing touch input graphical objects on the modules display. The controls are created using paramaters passed in the “new number-edit control object” subcommand, and style settings set by command 40.

The number-edit control supports enabled and disabled states, and has configurable low and high values. Each of the states has configurable colors which can be changed at any time.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Number-Edit Control Object / Change Option Flags

New Number-Edit Control Object

This command creates a new touch-screen operated number edit control graphical object on the display.

It’s various style options (colors, etc) can be set using command 40 (see the list below).
All style preference are adopted by the new object at time of creation. After creation, style options are unique to the object.

Option flags, bit5 enables position report packets to be sent to the host on value change. This differs from the “touch report options” which report object touch locations only.

Style options used are:

- Number-Edit opacity = Style: Object Opacity
- Number-Edit label text style = Style: True-Type Font Slot Selection
- Number-Edit label text size = Style: True-Type Font Size
- Number-Edit corner radius = Style: Corner Radius
- Enabled-state Background fill color = Style: Fill-A Color
- Enabled-state Border color = Style: Border-A Color
- Enabled-state Text label color = Style: Text-A Color
- Enabled-state Text value color = Style: Text-B Color
- Enabled-state Button color = Style: Control-A Color
- Disabled-state Background fill color = Style: Fill-C Color
- Disabled-state Border color = Style: Border-C Color
- Disabled-state Text label color = Style: Text-C Color
- Disabled-state Text value color = Style: Text-C Color
- Disabled-state Button color = Style: Control-C Color

Command Packet:

type:	49 (0x31)
data_length:	20 to 124
data[0]:	0 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = no background fill bit2 = label position (0=left, 1=right) bit3 = is a percentage (adds % on value) bit4 = UTF-16 text labels bit5 = report packet on value change (see below)
data[3-4]:	x position (16bit, signed, little-endian)

data[5-6]:	y position (16bit, signed, little-endian)
data[7-8]:	width (16bit, unsigned, little-endian)
data[9-10]:	height (16bit, unsigned, little-endian)
data[11]:	z-index
data[12]:	touch report option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[13-14]:	minimum value (16bit, signed, little-endian)
data[15-16]:	maximum value (16bit, signed, little-endian)
data[17-18]:	starting value (16bit, signed, little-endian)
data[19+]:	text label

Successful Return Packet:

type:	113 (0x31 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Report Packet:

type:	136 (0x08 | 0x80)
data_length:	2
data[0]:	number-edit object ID number
data[1-2]:	updated value (16bit, signed, little-endian)

Change Number-Edit Control Option Flags

Use this command to update the option flags after the control has been created.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type:	49 (0x31)
data_length:	3
data[0]:	0 (subcommand number)
data[1]:	existing number-edit control object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = no background fill bit2 = label position (0=left, 1=right) bit3 = is a percentage (adds % on value) bit4 = UTF-16 text labels bit5 = report packet on value change

Successful Return Packet:

type:	113 (0x31 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Set / Read Number-Edit Control Value

Set Number-Edit Control Value

Update the number-edit control value.

Command Packet:

type:	49 (0x31)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	existing number-edit control object ID number (1 to 255 valid)
data[2-3]:	new value (16bit, signed, little-endian)

Successful Return Packet:

type:	113 (0x31 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Read Number-Edit Control Value

Read the number-edit control value.

Command Packet:

type:	49 (0x31)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	existing number-edit control object ID number (1 to 255 valid)

Successful Return Packet:

type:	113 (0x31 | 0x40)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	number-edit control object ID number
data[2-3]:	current value (16bit, signed, little-endian)

50 (0x32) Checkbox Control Command Group

This command group allows the user to create simple checkbox touch input graphical objects on the modules display. The controls are created using paramaters passed in the “new checkbox control object” subcommand, and style settings set by command 40.

The checkbox control supports un-checked, checked, crossed and disabled states. Each of the states has configurable colors, which can be changed at any time.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Checkbox Control Object / Change Option Flags

New Checkbox Control Object

This command creates a new touch-screen operated checkbox control graphical object on the display.

It’s various style options (colors, etc) can be set using command 40 (see the list below).
All style preference are adopted by the new object at time of creation. After creation, style options are unique to the object.

Option flags, bit6 enables position report packets to be sent to the host on checkbox state change. This differs from the “touch report options” which report object touch locations only.

Style options used are:

- Checkbox opacity = Style: Object Opacity
- Checkbox label text style = Style: True-Type Font Slot Selection
- Checkbox label text size = Style: True-Type Font Size
- Checkbox corner radius = Style: Corner Radius
- Enabled-state Background fill color = Style: Fill-A Color
- Enabled-state Border color = Style: Border-A Color
- Enabled-state Text label color = Style: Text-A Color
- Enabled-state Text value color = Style: Text-B Color
- Enabled-state Button color = Style: Control-A Color
- Disabled-state Background fill color = Style: Fill-C Color
- Disabled-state Border color = Style: Border-C Color
- Disabled-state Text label color = Style: Text-C Color
- Disabled-state Text value color = Style: Text-C Color
- Disabled-state Button color = Style: Control-C Color

Command Packet:

type:	50 (0x32)
data_length:	15 to 124
data[0]:	0 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = Disabled state bit1 = No background fill bit2 = Label position (0=left, 1=right) bit3 = Tick state enabled bit4 = Cross state enabled bit5 = UTF-16 text labels bit6 = report packet on value change (see below)

data[3-4]:	x position (16bit, signed, little-endian)
data[5-6]:	y position (16bit, signed, little-endian)
data[7-8]:	width (16bit, unsigned, little-endian)
data[9-10]:	height (16bit, unsigned, little-endian)
data[11]:	z-index
data[12]:	touch report option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[13]:	starting state: 0 = none 1 = ticked 2 = crossed
data[14+]:	text label

Successful Return Packet:

type:	114 (0x32 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Report Packet:

type:	137 (0x09 | 0x80)
data_length:	2
data[0]:	checkbox object ID number
data[1]:	updated state: 0 = none 1 = ticked 2 = crossed

Change Checkbox Control Option Flags

Use this command to update the option flags after the control has been created.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type:	50 (0x32)
data_length:	3
data[0]:	0 (subcommand number)
data[1]:	existing Checkbox control object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = Disabled state bit1 = No background fill bit2 = Label position (0=left, 1=right) bit3 = Tick state enabled bit4 = Cross state enabled bit5 = UTF-16 text labels bit6 = report packet on value change (see below)

Successful Return Packet:

type:	114 (0x32 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Set / Read Checkbox Control Value

Set Checkbox Control Value

Update the checkbox control value.

Command Packet:

type:	50 (0x32)
data_length:	3
data[0]:	1 (subcommand number)
data[1]:	existing checkbox control object ID number (1 to 255 valid)
data[2]:	new state: 0 = none 1 = ticked 2 = crossed

Successful Return Packet:

type:	114 (0x32 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Read Checkbox Control Value

Read the checkbox control value.

Command Packet:

type:	50 (0x32)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	existing checkbox control object ID number (1 to 255 valid)

Successful Return Packet:

type:	114 (0x32 | 0x40)
data_length:	3
data[0]:	1 (subcommand number)
data[1]:	Checkbox control object ID number
data[2]:	current state: 0 = none 1 = ticked 2 = crossed

51 (0x33) Progress Bar Command Group

This command group allows the user to create progress bar graphical objects on the modules display. The progress bar is created using paramaters passed in the “new progress bar object” subcommand, and style settings set by command 40.

The progress bar is not used for user-input via the touch panel. It is only a value/status display object. It does support enabled and disabled states.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Subcommand 0 - New Progress Bar Object / Change Option Flags

New Progress Bar Object

This command creates a new progress bar graphical object on the display.

It’s various style options (colors, etc) can be set using command 40 (see the list below).
All style preference are adopted by the new object at time of creation. After creation, style options are unique to the object.

Style options used are:

- Progress Bar opacity = Style: Object Opacity
- Progress Bar label text style = Style: True-Type Font Slot Selection
- Progress Bar label text size = Style: True-Type Font Size
- Progress Bar corner radius = Style: Corner Radius
- Enabled-state Background fill color = Style: Fill-A Color
- Enabled-state Border color = Style: Border-A Color
- Enabled-state Text label color = Style: Text-A Color
- Enabled-state Text value color = Style: Text-B Color
- Enabled-state Bar background color = Style: Control-A Color
- Enabled-state Bar color = Style: Control-B Color
- Disabled-state Background fill color = Style: Fill-C Color
- Disabled-state Border color = Style: Border-C Color
- Disabled-state Text label color = Style: Text-C Color
- Disabled-state Text value color = Style: Text-C Color
- Disabled-state Bar color = Style: Control-C Color
- Disabled-state Bar background color = Style: Control-C Color

Command Packet:

type:	51 (0x33)
data_length:	20 to 124
data[0]:	0 (subcommand number)
data[1]:	new object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = no background fill bit2 = is a percentage value (adds % on value) bit3 = value is on the right bit4 = value is on the left bit5 = UTF-16 text label
data[3-4]:	x position (16bit, signed, little-endian)

data[5-6]:	y position (16bit, signed, little-endian)
data[7-8]:	width (16bit, unsigned, little-endian)
data[9-10]:	height (16bit, unsigned, little-endian)
data[11]:	z-index
data[12]:	touch report option: 0 = off 1 = report single touch & release (including position) 2 = report single touch, position updates and release
data[13-14]:	minimum value (16bit, signed, little-endian)
data[15-16]:	maximum value (16bit, signed, little-endian)
data[17-18]:	starting value (16bit, signed, little-endian)
data[19+]:	text label

Successful Return Packet:

type:	115 (0x33 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Change Progress Bar Option Flags

Use this command to update the option flags after the control has been created.

See Graphics Core Command Group, Command 40, for additional graphics object changes such as style options, (x,y) coordinates, z-index changes, etc.

Command Packet:

type:	51 (0x33)
data_length:	3
data[0]:	0 (subcommand number)
data[1]:	existing progress bar object ID number (1 to 255 valid)
data[2]:	option flags: bit0 = disabled state bit1 = no background fill bit2 = is a percentage value (adds % on value) bit3 = value is on the right bit4 = value is on the left bit5 = UTF-16 text label

Successful Return Packet:

type:	115 (0x33 | 0x40)
data_length:	1
data[0]:	0 (subcommand number)

Subcommand 1 - Set / Read Progress Bar Value

Set Progress Bar Value

Set the progress bar value.

Command Packet:

type:	51 (0x33)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	existing progress bar object ID number (1 to 255 valid)
data[2-3]:	new value (16bit, signed, little-endian)

Successful Return Packet:

type:	115 (0x33 | 0x40)
data_length:	1
data[0]:	1 (subcommand number)

Read Progress Bar Value

Read the progress bar value.

Command Packet:

type:	51 (0x33)
data_length:	2
data[0]:	1 (subcommand number)
data[1]:	existing progress bar object ID number (1 to 255 valid)

Successful Return Packet:

type:	115 (0x33 | 0x40)
data_length:	4
data[0]:	1 (subcommand number)
data[1]:	progress bar object ID number
data[2-3]:	current value (16bit, signed, little-endian)

62 (0x3E) Debugging

This command provides access to the modules internal debugging / logging output.

If enabled, debugging report packets are sent on the USB and serial interface if they are enabled, and have the packet interpreter enabled. The report packets have the command/reply number of 190. Long debug output strings are truncated to 124 bytes in length. If using cfTest, make sure the “Debugging” checkbox under the packet viewer is also enabled.

The “USB Debug” output option uses a second virtual USB COM port which needs to be enabled by holding the Left Key and Cross Key during module power-on/reset, or by a file called “CDCDEBUG_MODE” existing on the on board flash during power-on/reset. The output is color encoded using ANSI escape codes, so a compatible serial console viewer must be used (for example, PuTTY on Windows, or GTKTerm on Linux).

The “Serial Debug” output option uses Pin 2 on H1, but only if the serial interface is enabled, and the packet interpretor is disabled (see command 33). The output is color encoded using ANSI escape codes, so a compatible serial console viewer must be used (for example, PuTTY on Windows, or GTKTerm on Linux).

Note:
The debug levels set by this command are not saved by command 4. The debug output options must be set after every module power-cycle / reset.

Command Packet:

type:	62 (0x3E)
data_length:	3
data[0]:	USB/serial report packet level: 0 = disabled (default) 1 = error level 2 = warning level (and above) 3 = notice level (and above) 4 = debug low (and above) 5 = debug high (and above)
data[1]:	USB debug level:

(level options as above)
data[2]: serial debug leve:
(level options as above)

Successful Return Packet:

type: 126 (0x3E | 0x40)
data_length: 0
data[]: N/A

Report Packet:

type: 190 (0x3E | 0x80)
data_length: 0 to 124
data[]: ASCII debugging output text