CAIRN USB LED INTERFACE MANUAL

VERSION DATED 4 May 2022

INTRODUCTION

This interfacing system is potentially available in both two-channel and four-channel versions, starting with a two-channel version, in the form of a USB-controlled OptoLED. The USB interface, the design and operation of which has its roots in the USB interfacing for our now well-established Optospin filter wheel, allows both digital and analog control of the OptoLED's primary functions. However, all the existing hardware controls of the standard OptoLED have been retained as well, therefore offering a wide variety of control options. This also means that use of the USB control facilities is optional rather than essential. It will potentially control both the full OptoLED and the more recent versions of the OptoLED "Lite", the improvements to which have been such as to make it a more cost-effective product under most circumstances (the main remaining potential advantage of the full OptoLED being its ability to drive LEDs that consist of multiple chips in series, but we're trying to move away from these now).

To this we can now add two further possibilities. One of these is a four-channel OptoLED, which will effectively be two OptoLEDs in a single box, but with the USB interface controlling all four channels. The other is a possible version of our new dedicated four-channel MultiLED, which in its simplest version has no USB interface, with instead a single common set of inputs that together drive four all four LEDs together, but with the possibility in hardware of subsequently varying the levels of the individual LEDs, and/or of switching them on and off. The proposed USB version of this product will provide the same control facilities as for the USB OptoLEDs, by taking full advantage of the hardware facilities just mentioned.

LED HARDWARE OPTIONS

Previously we have tended to offer just one and two-channel LED systems, but with this interface being a four-channel one, we have been paying greater attention to the hardware aspects of using up to four LEDs together. Details are described elsewhere, but the outline possibilities are as follows.

First of all, our dedicated two-channel TwinLED and FuraLED hardware products will continue to be available, and the USB interface will be able to support them in exactly the same way as for using individual LED heads. These will now be joined by our dedicated four-channel MultiLED head, which is primarily going to be offered as a low-cost "white" light source with four fixed LED wavelengths, but for which a USB interface version will also be potentially available.

However, for fuller flexibility, we shall of course be continuing to offer our standard

LED heads, so that people can choose whatever wavelengths they like. For this option we have now also addressed the problem that our standard LED coupling system, although very versatile, does become rather bulky and expensive when in a full four-channel form in particular. Therefore we can now offer a more compact and cost-effective solution for this type of application – just ask for further details.

DUAL-DAC ARCHITECTURE

A key feature of the USB interface is its "dual DAC" (digital to analog converter) architecture. One of these DACs (the "level" DAC) allows the LED control current to be set optionally via a USB command, while nevertheless retaining the standard OptoLED's options of control via a front panel potentiometer or an external voltage. The other acts as a USB-programmable potentiometer that can automatically set the full scale control current, from whatever source, to match the maximum safe current for the LED that it is actually driving. This is a backwards-compatible feature, as it works with any of our LEDs. However, since our LED protection circuitry does allow transient overdriving, a USB command can double the full scale current, up to the maximum achievable value of 5A, in which case the LED's protection circuitry will of course come into operation if the overdriving persists for too long.

Both DACs are 12 bit resolution, giving a potential resolution of 1 part in 4096, but scaled so that internally a "full scale" value is actually 4000. This resolution of 1 part in 4000 is available if required, but for greater user-friendliness the default operating mode is to specify values as percentages of full scale with 0.1% resolution.

However, still greater resolution can be achieved at lower control ranges! The resolution of the "scale" DAC, which sets the full scale current for that particular LED, is sufficient to allow a "low scale" mode, in which its setting is reduced by a factor of ten, so that the level DAC settings now allow currents of up to 10% of full scale to be set at 0.01% resolution. This works for the manual control and external input signals as well as for the USB level-setting command.

And on (future) LED heads with optical feedback, a "HIGAIN" command is available to increase the optical feedback gain by a factor of ten, which has the same effect. These two features can be combined to give levels of up to 1% of full scale with 0.001% resolution. Note that we say "levels" rather than "currents" here, since the luminous output from an LED isn't absolutely linear with current, whereas optical feedback does linearise the optical response.

OPTICAL FEEDBACK

In designing the USB interface it became clear that the decision whether to use optical feedback or not is best determined by the LED heads themselves. The proposed future protocol for our USB LED products is that optical feedback is always in use if available, and its presence is determined automatically by our existing

connector system, so this is transparent to the user. However, if the user does wish to switch feedback on or off in the cases where it is available, this can be done in principle by incorporating a switch within the LED head, rather than on the rear panel as currently. Therefore the USB interface does no need to know whether optical feedback is in operation or not, except that optical feedback must be operating for the "HIGAIN" mode to be available, and that appropriately fitted LED heads will provide this information to the system.

DIGITAL CONTROL OPTIONS

The USB control also allows the LEDs to be controlled digitally, but here too the existing OptoLED system is retained, in the form of the on/auto/off front panel switch and the two external digital control inputs for each channel. (In our MultiLED products there will be just one set of such inputs that will control all four channels together, but the USB interface will allow independent channel control beyond that point.) To this the USB interface potentially adds its own "digital off" control signal, which overrides the front panel switch setting. In the absence of that control signal, for an LED to be digitally on, the front panel switch must itself either be in the "on" position, or in the "auto" position with an external digital control signal present. However, the "LED ON" indicator will also be switched off by the USB interface's "digital off" control signal, so it will continue to be a reliable indication of whether a given LED is actually on or not. By default, the digital control signal from the USB interface will be "on" at switchon, so that the unit can be used without needing to be connected to a computer, although any other configuration can be set up by the various USB commands and then stored by the SAVE_CONFIGURATION command instead if preferred.

NEW ALPHANUMERIC DISPLAY

A potentially very useful further addition to the USB interface is an alphanumeric display, which is available in both two- and four-channel versions, nicely matching or proposed USB product versions. For each channel it clearly shows whether it is on or off according to USB control, and in all modes it shows how hard the LED is being driven, both as a percentage of the full scale (as set by the "scale" DAC as previously described, so that 100% corresponds to the maximum "safe" current for an LED) and as the calculated corresponding current. This alone marks a significant improvement over the simple current meters that we've been using until now, but the display can show many other things besides, as we now describe.

LED HEAD NONVOLATILE MEMORY

Our future LED heads will incorporate a nonvolatile memory chip for recording a variety of data associated with the LED, and the USB interface allows parameters such as wavelength, maximum drive current, and the specification of any associated optical filters, to be recorded or saved by this means. As well as the stored

information for these parameters being retrievable by the USB interface, they are also directly shown on the display, which is a particularly user-friendly feature. Although not being such a useful parameter for direct display, the "scale" DAC setting for determining the maximum LED current is a particularly useful one, as once programmed it will continue to be available indefinitely. Note that this memory chip can also be retrofitted to any of our existing LED heads if required, as it's a pretty straightforward operation.

SYSTEM NONVOLATILE MEMORY

We can use this facility in two different ways. First, if an LED head doesn't have its own memory chip, exactly the same parameters can be stored in system memory instead. This happens automatically if there is no LED head memory. Clearly the information will only remain correct for as long as that particular LED head remains connected, but in practice LED heads may not be interchanged so often, although of course storing the information there is the better option. If an LED head with its own memory is subsequently fitted instead, then its contents will be used in preference to whatever may have been stored in system memory, so this doesn't introduce any conflicts.

The second way in which this facility can be used is to store the system configuration, as we shall now describe.

AUTOCALIBRATION AND SYSTEM CONFIGURATION

The new facility for automatically setting the full-scale input level to match the maximum safe current for a given LED is going to be a welcome development for many users. In LED heads with memory chips, these will be supplied with this parameter already programmed, but it will also be available for LED heads without them, as here too this information can be stored in programmable memory within the interface. The autocalibration process is very straightforward, as it consists of steadily ramping up the LED current until the LED head's protection circuitry causes it to cut out. The full-scale input level is then set to be approximately 10% below this value (see the section describing the boost facility if you want to drive the LED any harder).

It can be run at any time for all connected LEDs by pressing the RESET button on the rear of the unit. It can be prevented from running for a given LED by keeping the front panel switch in the "off" position. Alternatively, it can be run at any time by the CALIBRATE_FULL_SCALE command. The reset also restores the system configuration to its default condition, in which the USB interface leaves the system under full manual control, so that it doesn't need to be connected to a computer in order to run. However, some users may prefer to have the system to start in some other configuration, such as being digitally switched off by USB, and/or with some specific current level programmed under USB control. In that case, once a particular

configuration has been set up (and which includes the programmable timer facility, to be described next), it can be saved to system memory for future use by the USB "SAVE CONFIGURATION" command. This combination of facilities should give users the best of all worlds.

DIGITAL INTERFACE AND PROGRAMMABLE TIMERS

A comprehensive digital interface is now included. As well as individual digital outputs corresponding to the status of the up to four LEDs, these are supplemented by a similar number of "auxiliary" outputs, and all of them are programmable. There are also four digital inputs, to which the programmable events can be synchronised. One of these will typically be used as an "expose" input to indicate the start of a camera acquisition. Each digital output can be programmed to have (up to) two "on" and two "off" events following receipt of an "expose" signal, at time delays of up to 10 seconds but with millisecond resolution and a precision of just a few (less than 10) microseconds. The LEDs will follow the status of the first four, and the other four can be put to any other required use, such as mechanical changing of optical filters.

Alternatively, the timers can be operated in a "freerunning" mode, where they will cycle continuously for a programmable period of up to ten seconds. This is now all running very nicely, and further facilities could easily be added, according to whatever suggestions anyone may have.

After any timer-related command is entered, the alphanumeric display will change to show the programmed times, so that the effects of the command can be clearly seen. The display will revert to its normal mode as soon as any further USB command is entered. It will also show the programmed times whenever the timer code is running.

BOOST FACILITY

In our products with interchangeable LED heads, these will continue to have built-in protection networks that prevent them from being damaged when overdriven. However, since the maximum drive current for a given LED is now programmed by the setting of the "scale" DAC, they can no longer be overdriven in normal use. But more intrepid users can nevertheless choose to overdrive an LED if they so wish, by using the "boost" command. For all input modes (front panel control, external analog input or USB level control), the drive current for a given input setting is doubled, up to a maximum level of 5A. When this facility is in use, "x2Boost" will be shown in place of the text field on the display, unless it is limited by the 5A maximum, in which case "<2Boost" will be displayed instead, to indicate that the actual boost factor (which is a constant one for all input values) is somewhat less than two. In all cases the display will show the boosted percentage and current, and hence including the effect of a boost factor being reduced by the 5A limit.

The boost facility is only available on the normal current control range (and also not

available on the MultiLED version). It cannot be entered if the “low range” USB level option has been selected, or if the “HIGAIN” option for LED heads with switchable feedback gain has been selected. Selection of either of those options will cause the boost facility to be switched off if it was previously set. It is also not available on our MultiLED products, as here the maximum current levels will have been preset in hardware according the (noninterchangeable) individual LEDs in the head assembly.

USB COMMAND INTERFACE

The USB command interface works in the same way as it does in the Optospin. Each command has a two-byte identifier, which may be followed by one or more further bytes. For example, many commands can act on any of the up to four channels that the interface supports, in which case a “channel identifier” byte is required, and there may then be further data bytes to send. If a command is executed successfully, it returns a “true” (0FFh) byte, always followed by another byte that itself gives the number of subsequent bytes that are being returned, although that may often be zero. If it fails for any reason, then only two bytes are returned, the first being zero to indicate failure, and the second being an error number, although that is also generally zero in this application. If a command is only “partially” successful, which usually relates to whether data that would normally be stored in LED head memory if present is instead stored in system memory, then a byte value of 1 is returned as the first byte rather than 0FFh or zero.

Please note that where, as in most cases, a command takes a “channel identifier” byte, values outside the permissible range of 0-3 will return a standard “zero” error code. However, at least as currently implemented, the software doesn’t check whether a channel is physically present or not, so sending a command to channels 2 or 3 of a two-channel OptoLED won’t come back with an error. I could in principle check for this, and maybe return with a different error code, but on the other hand the host computer has the GET_LED_CHANNELS_PRESENT command at its disposal, so it can always check this situation for itself.

The original reason for the two-byte command identifier is that in the Optospin the number of commands is sufficiently large to require a second command “page”, hence with a first byte of either 0 or 1, and we decided to retain the same format here. However, the number of USB commands for this application has now also recently expanded to need the second page, so that decision has saved us a few problems!

MARCH 2022 UPDATE

The command list is believed to be up to date as of today, but some additions or changes may nevertheless occur!

GET_VERSION (00 40h) (-- n1 n2)

This returns two bytes to indicate the current software version, major byte first.

INITIALISE_USB (00 44h) (--)
This just resets and generally cleans out the USB interface in case any data has been left in the data buffer for any reason.

SWITCH_LED_ON (00 48h) (n1 --)
This command is followed by a single byte for channel selection. Values of 0-3 switch on LED channels 1-4 respectively. Values higher than 3 (0FFh recommended) will switch on all channels. Note for an LED to actually be on, the on/off/auto front panel switch needs either to be in the "on" position, or in the "auto" position and with an external digital command signal. The default situation is "on" at switchon, so that no USB commands need to be sent in order to use the unit.

SWITCH_LED_OFF (00 4Ch) (n1 --)
This command is followed by a single byte for channel selection. Values of 0-3 switch off LED channels 1-4 respectively. Values higher than 3 (0FFh recommended) will switch off all channels. Note that this command will switch an LED off regardless of the position of the on/off/auto front panel switch.

GET_LED_ON/OFF (00 50h) (n1 -- n2 n3)
This command is followed by a single byte for channel selection (values 0-3 for channels 1-4). It returns two values, both of which will be 0FFh if the LED is on. If the LED is switched off by the USB command, the first byte will be zero. If the LED is switched off at the front panel (or if that switch is in the auto position but with no digital input), then the second byte will be zero. (Command updated 30 September 2020)

SWITCH_USBV_ON (00 54h) (n1--)
This command is followed by a single byte for channel selection (values 0-3 for channels 1-4). It selects the USB-programmed LED drive level (see SET_USB_LEVEL) for that channel, regardless of the panel control settings. Values above 3 (0FFh is recommended) select this for all channels. The default situation is "off" at switchon, so that no USB commands need to be sent in order to use the unit.

SWITCH_USBV_OFF (00 58h) (n1--)
This command is followed by a single byte for channel selection (values 0-3 for channels 1-4). It deselects the USB-programmed LED drive level (see SET_USB_LEVEL) for that channel, which is now determined by the panel control settings. Values above 3 (0FFh is recommended) deselect all channels.

GET_USBV_ON (00 5Ch) (n1--n2)
This command is followed by a single byte for channel selection. It returns a value of 0 if the front panel control or external voltage input is driving that LED channel, or 0FFh it is being determined by the SET_USB_LEVEL value.

SWITCH_HIGAIN_ON (00 60h) (n1 --)
This command only works with LED heads that offer optical feedback with switchable gain. The effect of the feedback gain switching is to reduce the LED current for a given drive signal by a factor of ten. It is followed by a single byte for channel selection (values 0-3 for channels 1-4). The default situation is “off” at switchon.

SWITCH_HIGAIN_OFF (00 64h) (n1 --)
This command is the counterpart to SWITCH_HIGAIN_ON, and is followed by a single byte for channel selection (values 0-3 for channels 1-4).

GET_HIGAIN (00 68h) (n1—n2 n3)
This command is also followed by a single byte for channel selection. Feb2022 it now returns two byes. The first byte is 0FFh if the HIGAIN option is available, otherwise 0. The second byte is 0FF if HIGAIN is on, otherwise 0.

It returns a value of zero in “normal” mode, or 0FFh if the HIGAIN mode has been selected.

SET_USB_LEVEL (00 6Ch) (n1 n2 n3 --)
This command is followed by 0-3 for the channel number, and then by two further bytes for the percentage level. The first byte is 0-100, and the second is 0-9 for the fractional percentage, which will be interpreted as zero if the first byte is 100

GET_USB_LEVEL (00 70h) (n1 -- n2 n3)
This command is followed by 0-3 for the channel, and then by two further bytes representing the stored percentage level in the same format as for SET_USB_LEVEL. If the stored level was at the full 12bit resolution (0-4095) supported by SET-12BIT_USB_LEVEL, the nearest (lowest) equivalent percentage values will be returned, and the initial acknowledgement byte will just be 1 rather than 0FFh in order to indicate this. In that case the actual value can be read by GET_12BIT_USB_LEVEL.

SET_12BIT_USB_LEVEL (00 74h) (n1 n2 n3 --)
This command is followed by 0-3 for the channel, and then by two further bytes in 16bit format, high byte first, with the four high-order bits in this byte being interpreted as zero. Note that this permits the full converter range of 0-4095 to be set, but the internal calibration is such that 4000 represents full scale, so higher numbers will give a correspondingly higher signal value, although this will be correctly dealt with by the system hardware.

GET_12BIT_USB_LEVEL (00 78h) (n1 -- n2 n3)
This command is followed by 0-3 for the channel, and it returns two bytes in the same 16bit format as for SET_12BIT_USB_LEVEL. If the level had actually been set by SET_USB_LEVEL, then the 12bit equivalent value, which will always be

exact, will be correctly returned in 16bit format.

SET_DECIMAL_SCALE (00 7Ch) (n1 n2 n3 --)
(Note that most users will probably never need to use this command, as in practice it would be preferable to set the scale automatically by the CALIBRATE_FULL_SCALE command.) This command is followed by 0-3 for the channel, and then by two further bytes for the scale value in the range 0-400. The high byte (0-4) sets the hundreds, and the low byte (0-99) sets the tens and the units. It is used to set the scale DAC value for that LED channel, where a value of 400 as entered corresponds to no attenuation, so that a 5V input signal will give the full scale maximum LED current of 5A. The values as entered are multiplied by ten, so that those stored in the DAC are actually 0-4000. By ensuring that the stored values are multiples of ten, this allows the SET_LOW_SCALE command to set an exact tenfold attenuation when it is executed. However, the full potential 12bit resolution is nevertheless achievable by the alternative SET_12BIT_SCALE command if required. Also note that execution of this command will put that channel back into normal scale mode if it had been running in low scale mode. It will also cancel boost mode if that had previously been selected. Values entered by this command are stored in system memory for future use, as long as they are greater than 25 (lower values are unlikely to be practical for future use). LED head memory, if present, is NOT updated – see CALIBRATE_FULL_SCALE for further information.

GET_DECIMAL_SCALE (00 80h) (n1 -- n2 n3)
This command is followed by 0-3 for the channel, and returns the same two number format that is used for SET_DECIMAL_SCALE, again with the high (hundreds) byte first. If the stored (12bit) value was not a precise multiple of ten, then the next lowest one that could have been set by SET_DECIMAL_SCALE is returned, and the first acknowledgement byte is 1 instead of 0FFh in order to indicate this. Note that the numbers returned remain the same, regardless of the whether that channel is in normal or low scale mode, even though in low scale mode the actual number stored in its scale DAC will have been reduced by a factor of ten. It will also cancel boost mode if that had previously been selected.

SET_12BIT_SCALE (00 84h) (n1 n2 n3 --)
This command is followed by 0-3 for the channel and then by two further bytes in 16bit format, high byte first, with the four high-order bits in this byte being interpreted as zero. Note that this permits the full converter range of 0-4095 to be set, but the internal calibration is such that 4000 represents full scale, so higher numbers will give a correspondingly higher signal gain, although this will be correctly dealt with by the system hardware. Also note that execution of this command will put that channel back into normal scale mode if it had been running in low scale mode. It will also cancel boost mode if that had previously been selected. Values entered by this command are stored in system memory for future use, as long as they are greater than 255 (lower values are unlikely to be practical for future use). LED head memory, if present, is NOT updated – see CALIBRATE_FULL_SCALE for further information.

GET_12BIT_SCALE (00 88h) (n1-- n2 n3)
This command is followed by 0-3 for the channel, and it returns two bytes in the same 16bit format as for SET_12BIT_SCALE. If the level had actually been set by SET_DECIMAL_SCALE, then the 12bit equivalent value, which will always be an exact multiple of ten, will be correctly returned in 16bit format. It will also cancel boost mode if that had previously been selected.

SET_LOW_SCALE (00 8Ch) (n1 --)
This command is followed by 0-3 for the channel. It reduces the system gain, set by the scale DAC, by a factor of ten, so all inputs, from whatever source, are reduced by this same factor. It means that for all inputs, instead of being from 0.0 to 100.0% of full scale, they are now 0.00% to 10.00% of full scale. Note that for guaranteed correct operation of this command, the scale DAC value should have been set by SET_DECIMAL_SCALE or CALIBRATE_FULL_SCALE, rather than by SET_12BIT_SCALE, as this will ensure that the scale DAC value can be reduced by the exact factor of ten required for the correct operation of this command. If not, the nearest lowest scale DAC value is used, and the first acknowledgement byte is 1 rather than 0FFh to indicate this. It will also cancel boost mode if that had previously been selected.

SET_NORMAL_SCALE (00 90h) (n1 --)
This command is followed by 0-3 for the channel. It restores the scale DAC to its normal value, as was at some time previously set (preferably) by SET_DECIMAL_SCALE, or by SET_12BIT_SCALE, so that inputs are now (again) 0.0 to 100.0% of full scale.

GET_WHICH_SCALE (00 94h) (n1 -- n2)
This command is followed by 0-3 for the channel. It returns 0FFh if that channel is running at normal scale, and 0 if it is running in the tenfold lower low scale mode.

GET_INPUT_LEVEL (00 98h) (n1-- n2 n3)
This command is followed by a channel identifier byte (values 0-3), and returns the input signal level, from whichever source it has been set (set by the front panel control, external input voltage or USB command voltage) in the form of two bytes representing a 10-bit number, high byte first. The decimal range is 0-1023, scaled such that 1,000 represents the full scale input level of 5V. For all possible inputs (front panel, external or USB level) the level is derived from the analog-to-digital conversion circuitry, so if it's the USB level that's of specific interest, then it makes more sense to use the dedicated commands for that.

CALIBRATE_FULL_SCALE (00 9Ch) (n1 -- n2)
This action shouldn't ordinarily be necessary, as the calibration information should already be stored in the LED head if it has a memory chip, or in the system box for that LED channel if not. However, as long as the front panel switch is in the "on"

position, it can be carried out at any time by executing this command. It is followed by a byte (value 0-3) to select the channel, and returns 0FFh if it was run successfully, or 0 otherwise (eg if the LED is still switched off by the panel controls). A successful calibration will also clear boost mode if it was previously selected, and will also select normal scale mode if low scale mode was previously selected. The result will also be stored in the system box for future use, although if the LED head has a memory chip the value stored there will always be used instead. If the LED head memory is to be updated, this can then be done by the STORE_SCALE_TO_LED_HEAD command.

SET_BOOST_MODE (00 A0h) (n1 –)
(Not available for the MultiLED, as that doesn't support transient overdriving!)
The calibration procedures ensure that an LED cannot normally be overdriven from either the USB or the front panel controls, or by a full scale (5V) external input, but its protection network does generally allow higher currents to be passed for short periods. This command, which is followed by a byte (value 0-3) to select the channel, doubles the maximum current for all modes, subject to the general maximum limit of 5A, in order to allow transient overdriving. Note that boost mode is not available in the "HIGAIN" and "Low Range" modes, and selection of either of those modes will cancel boost mode if it was previously set. It is also switched off by a successful full scale calibration procedure.

CLEAR_BOOST_MODE (00 A4h) (n1--)
This command is the counterpart of SET_BOOST_MODE, and works in exactly the same way.

GET_BOOST_MODE (00 A8h) (n1 -- n2)
This command is followed by the channel number (0-3), and returns 0FFh if that channel is being boosted, or 0 otherwise. Input values outside this range will return an error.

GET_LED_CHANNELS_PRESENT (00 ACh) (-- n1)
This returns a single byte, with bits 0-3 set if LED channels 0-3 respectively are physically present. We are likely to be selling both two-channel and four-channel USB systems, and this command lets the controlling software know which if required.

SAVE_CONFIGURATION (00 B0h) (--)
The current operating configuration, as set by the various other USB commands, is stored in system memory, making it immediately available once again for future use.

RESET_CONFIGURATION (00 B4h) (--)
This is the software equivalent of pressing the RESET button, and does just what it says. Amongst other uses, it ensures that a unit that may have been programmed in such a way as to require a USB connection can be returned to its default condition in

which the USB interface is effectively inactive, allowing the unit to be operated by its own controls and switches. The same as with pressing the RESET button, any previously set configuration will be preserved for future use, but this default condition will be preserved if the SAVE_CONFIGURATION command is then executed.

SET_LED_WAVELENGTH (00 B8h) (n1 n2 n3 --)
This command is followed by the channel number (0-3) and then by two further bytes which set the wavelength in nanometers for the alphanumeric display for that channel. The first byte (maximum of 9) sets the hundreds digit, and the second byte (0-99) sets the tens and the units. If the first byte is zero, then regardless of the second byte (although zero is recommended here too) the display will show “White”. If no valid wavelength has been set, the display will show “Wlen?” as a prompt to do so. In LEDs with a memory chip, the set value will be stored there for future recall, but in its absence it will be stored in the system memory for that channel instead. In that case the command will return 1 rather than 0FFh as the first acknowledgement byte.

GET_LED_WAVELENGTH (00 BCh) (n1 -- n2 n3)
This command allows the USB interface to “see” the displayed wavelength value. It is followed by the channel number, and the wavelength is returned in the same two-byte format in which it was entered. It will instead give an error message if no wavelength data had been programmed. If the wavelength information is present, but is from system rather than LED head memory, then the first acknowledgement byte is returned as 1 rather than 0FFh.

SET_LED_FILTER_DATA (00 C0h) (n1 n2 n3 n4 --)
This command is followed by the channel number (0-3) and then by three further bytes to specify the characteristics of any optical filter that may have been fitted. The first two bytes set the filter (centre) wavelength to be shown on the alphanumeric display, in the same format (hundreds first, then tens and units) as for that of the LED itself. If the first byte is zero, then regardless of the second and third bytes (although zero is recommended for these too), then the display will just show “NoFilter”. Otherwise the third byte sets the displayed bandwidth, in the range 1-99 nanometers, but also with two special cases. Entering zero will cause “SP”, to signify “short pass”, to be displayed instead, and entering a value of more than 99 will cause the albeit unlikely case of “LP”, to signify “long pass”, to be displayed instead. The data will be stored and acknowledged in the same way as for SET_LED_WAVELENGTH. If no valid filter data have been entered, the display will show “Filter?” as a prompt to do so. As with SET_LED_WAVELENGTH, in LEDs with a memory chip, the set value will be stored there for future recall, but in its absence it will be stored in the system memory for that channel instead. In that case the command will return 1 rather than 0FFh as the first acknowledgement byte.

GET_LED_FILTER_DATA (00 C4h) (n1 -- n2 n3 n4)

This command allows the USB interface to "see" the displayed filter data. It is followed by the channel number, and the wavelength and bandwidth are returned in the same three-byte format in which they were entered. It will instead give an error message if no filter data had been programmed. If the filter information is present, but is from system rather than LED head memory, then the first acknowledgement byte is returned as 1 rather than 0FFh.

SET_LED_TEXT (00 C8h) (n1 n2 n3 n4 n5 n6 n7 n8 n9 --)
This allows an eight-character text field to be programmed on the alphanumeric display. It could perhaps be used to show some specific information about the LED itself, but in case the user doesn't wish to do so or is too daunted by the format of this command, then if not programmed the display will instead just show "Channel1", "Channel2" etc in this field as appropriate. The more intrepid user will first enter the channel number (0-3), followed by the standard ASCII codes for the eight text characters (ie 32 for a blank space, 65-90 for capitals A-Z, 97-122 for lower case a-z and 48-57 for numbers 0-9). The same as for SET_LED_WAVELENGTH and SET_LED_FILTER_DATA, this information will be stored in LED head memory if available, otherwise it will be stored in system memory for that LED channel, in which case the first acknowledgement byte will be 1 rather than 0FFh.

GET_LED_TEXT (00 CCh) (n1 -- n2 n3 n4 n5 n6 n7 n8 n9)
This command allows the USB interface to "see" the displayed text string. It is followed by the channel number (0-3) and returns the eight text characters. It will instead give an error message if no text string had been programmed. The same as for other commands for this type, if a text string has been programmed but stored in system rather than LED head memory, the first acknowledgement byte will be 1 rather than 0FFh.

SET_LED_TIMER_ONE (00 D0h) (n1 n2 n3 n4 n5 --)
This command is followed by a channel identifier of 0-3 for LED Channels 1-4, and then by four further bytes. These are read in pairs to set the times first for an "on" event and then for an "off" event, where the "times" are those after receipt of a camera "exposure" pulse at which they occur. These times set those of the "on" and "off" events with millisecond resolution, over the range 0-9999msec, after receipt of that pulse (or after an internally generated "recycle pulse", see the SET_CYCLE_TIME command). For each byte pair, the first byte in the range 0-99 sets the number of hundreds of milliseconds, and the second byte, also in the range 0-99, sets the units component. If either byte is greater than 99 (0FFh recommended for both), then that timer event will not occur. It is also possible (and perhaps useful) for an "off" time (which is programmed second in the sequence for this command) to precede an "on" time (which is programmed first). If no further exposure pulse is received after the maximum programmable switching time of 9999msec when the timer code is running, then the digital outputs will remain indefinitely in the state set by the latest programmed event, but in all cases the sequence will be reset to time zero on receipt of the next exposure pulse. Also at the start of an exposure, the

outputs will by default continue in the same state as they were at the end of the previous one, rather than being reset, but they can be programmed to switch on or off at time zero (just like at any other time) if that type of behaviour is preferred.

GET_LED_TIMER_ONE (00 D4h) (n1-- n2 n3 n4 n5)
This command, to match SET_TIMER_ONE, also takes a channel number of 0-3, and returns the corresponding programmed times that had been set by that command. The only difference is that an unprogrammed time, however set, is always returned as 0FFh 0FFh.

CLEAR_LED_TIMER_ONE (00 D8h) (n1--)
Although any "on" or "off" time can be cleared by entering 0FFh 0FFh, this command is an easier way of clearing both.

SET_LED_TIMER_TWO (00 DCh) (n1 n2 n3 n4 n5 --)
This command is functionally identical to SET_TIMER_ONE, and has been split away from it only to simplify the programming of the timers facility. It allows a second "on" time and a second "off" time to be programmed for any of the (again 0-3) channels in a totally equivalent way to SET_TIMER_ONE, and the programmed times (which are again the times of "on" and "off" events after a receipt of the "exposure" pulse rather than absolute durations) can be interleaved with the timer one times in whatever way may be required. If this is felt to give a user too much choice, then its existence can simply be ignored.

GET_LED_TIMER_TWO (00 E0h) (n1-- n2 n3 n4 n5)
This command, to match SET_LED_TIMER_TWO, also takes a channel number of 0-3 and returns the corresponding programmed times that had been set by that command. The same as for timer one, the only difference from the SET command is that an unprogrammed time, however set, is always returned as 0FFh 0FFh.

CLEAR_LED_TIMER_TWO (00 E4h) (n1--)
This is again identical in effect to CLEAR_LED_TIMER_ONE

SET_AUX_TIMER_ONE (00 E8h) (n1 n2 n3 n4 n5 --)
As well as up to four LED channels, the USB interface also supports the same number of "auxiliary" channels, in the form of independently programmable digital outputs. This command is the auxiliary counterpart to SET_LED_TIMER _ONE.

GET_AUX_TIMER_ONE (00 ECh) (n1-- n2 n3 n4 n5)
The auxiliary counterpart to GET_LED_TIMER_ONE

CLEAR_AUX_TIMER_ONE (00 F0h) (n1--)
The auxiliary counterpart to CLEAR~LED_TIMER_ONE

SET_AUX_TIMER_TWO (00 F4h) (n1 n2 n3 n4 n5 --)

The auxiliary counterpart to SET_LED_TIMER_TWO

GET_AUX_TIMER_TWO (00 F8h) (n1-- n2 n3 n4 n5)
The auxiliary counterpart to GET_LED_TIMER_TWO

CLEAR_AUX_TIMER_TWO (00 FCh) (n1-- n2 n3 n4 n5)
The auxiliary counterpart to CLEAR_LED_TIMER_TWO

SET_CYCLE_TIME (01 00h) (n1 n2 --)
Instead of waiting for a camera pulse, the timer code can automatically recycle after an interval set by this command. Since it applies to all channels, there is no channel identifier byte, and it just takes two bytes to encode the time in the same 0-9999msec two-byte format as used for the other timer commands. Entering a valid time automatically sets this mode.

GET_CYCLE_TIME (01 04h) (-- n1 n2)
This command returns the cycle time in the same two-byte format used for setting and returning the other times. If no valid time has been stored, then 0FFh 0FFh will be returned.

CLEAR_CYCLE_TIME (01 08h) (--)
Clearing the cycle time reverts the system to wait for an external camera pulse before running each timer sequence.

DISPLAY_LED_TIMERS (01 0Ch) (–)
This command shows the LED timer settings on the alphanumeric display. The display will return to its normal mode on the execution of any other command, apart from GET_DIGITAL_STATUS when called via 200h.

DISPLAY_AUX_TIMERS (01 10h) (–)
This command is the counterpart to DISPLAY_LED_TIMERS, except that the auxiliary timer settings are shown instead.

RUN_TIMERS (01 14h (--)
This command does just that. The timer code will run until the STOP_TIMERS command is executed, although any other USB command (apart from GET_DIGITAL_STATUS when called via 200h) will work just as well. If a valid cycle time has been stored, the timer code will run repeatedly, otherwise it will wait for receipt of a camera pulse before running the sequence, which will repeat whenever a further pulse is received. A continuously high pulse will cause the sequence to be repeated as soon as the previous one has finished. While the timers are running, cairn-shaped annunciators (how sad) on the alphanumeric display flash on and off to mirror the state of the LED and auxiliary timer outputs.

STOP_TIMERS (01 18h) (--)

Yes it does.

SAVE_TIMERS (01 1Ch) (n1 --)
This command is followed by the channel number 0-3. Both the LED and the auxiliary times for the specified channel, plus the cycle time if set, are stored in system memory, making them immediately available for future use.

STORE_SCALE_TO_LED_HEAD (01 20h) (n1 --)
This command is followed by the channel number 0-3. It stores the currently set scale DAC value in the LED head EEPROM for long term future use. If the LED head has no EEPROM an error condition is returned.

SET_HIGAIN_AVAILABLE (01 24h) (n1 --) *(For Cairn use only!)*
This command is followed by the channel number 0-3. It indicates and stores in the LED head that the optical feedback gain is switchable under USB control (see SWITCH_HIGAIN_ON and SWITCH_HIGAIN off), and so of course should be executed only for LED heads that have this facility. If they do, they will already have been programmed by this command, so it is NOT for general use.

CLEAR_HIGAIN_AVAILABLE (01 28h) (n1 --) *(For Cairn use only!)*
This command is provided just in case SET_HIGAIN_AVAILABLE had been executed in error.

CLEAR_PIC_MEMORY (01 2Ch) (n1 --)
This command clears the system's programmable memory (but not that in the LED heads). It's basically intended to be a "factory reset" if anything untoward appears to be going on with the software.

ENABLE_DISPLAY_EVENT_TIMES (01 30h) (--)
When any timer events are set or cleared, this command causes the alphanumeric display to show either the new LED or auxiliary timer settings as appropriate, until the next USB command is issued. Although clearly potentially useful, this does slow down the response of the USB interface to future commands, hence it has been made optional.

DISABLE_DISPLAY_EVENT_TIMES (01 34h) (--)
This just reverses the effect of ENABLE_DISPLAY_EVENT_TIMES. But note that the specific DISPLAY_LED_TIMERS (01 0Ch) and DISPLAY_AUX_TIMERS (01 10h) will continue to send their data to the alphanumeric display.

GET_DISPLAY_EVENT_TIMES (01 38h) (-- n1)
This command returns 0FFh if the alphanumeric display will be updated to show updated LED or auxiliary timer settings when they are entered, or zero otherwise.

GET_DIGITAL_STATUS (01 3Ch) (-- n1 n2 n3 n4 n5 n6 n7 n8 n9)

This command returns nine bytes to show the overall digital system status. Within each byte, the least significant bit is corresponds to the lowest channel number (system channel 0, user channel1). Bits corresponding to absent channels are returned as zero. The byte sequence is as follows.

1. Front panel switch status (bit will be high if switch is on, or in auto position when switched on digitally).
2. USB on/off status (bit will be high if USB on/off command is on).
3. USB level control status (bit will be high if USB is setting the level).
4. LED timer outputs (bit will be high if LED timer output is on).
5. Aux timer outputs (bit will be high if aux timer output is on).
6. Low scale status (bit will be high if in low scale mode, also the corresponding bit in the high nibble will be set if the low scale is exactly so).
7. HIGAIN status (bit will be high if the LED is in this mode).
8. Boost mode status (bit will be high if in boost mode, also the corresponding bit in the high nibble will be set if the boost is less than a factor of two, in which case the LED current will be limited to the system maximum of 5A).
9. Trigger input status (bit will be high if that trigger input is digitally high).

GET_ADC_ANALOG_STATUS (01 40h) (-- n1 n2 n3 n4 n5 n6 n7 n8)

This command returns two bytes for each channel in turn, high byte first. It is intended for frequent monitoring purposes, especially when the signal level is being set by the front panel control or by an external input, in which case it needs to be measured by the system's ADC converter. For consistency, the ADC converter is also used to measure the signal level if it is actually under USB control. This converter has 10-bit resolution, giving values of 0-1023, where a full scale 5V input signal is returned as 1000, so the number returned is in this same format. If that channel isn't present, then zero is returned. Note that the signal levels actually sent to the LED, from whatever source, will be reduced by a factor of 10 in either the low scale or high gain modes, or by 100 if both are set. Also (but in the normal signal range only), the signal levels sent to the LED in boost mode will be doubled, up to a system maximum of 5A.

GET_USB_ANALOG_STATUS (01 44h) (-- n1... n16)

This command returns the signal levels currently set by the USB interface for each channel, regardless of whether they are being used to control the LEDs. In this case there are four bytes per channel, with the first two bytes (high byte first) encoded as 0-4095, with 4000 corresponding to a full scale 5V input signal. The second two bytes encode the actual boost factor if boost mode is operating. This is a hex number in the range 4000h-8000h, where 8000h corresponds to a factor of 2, and 4000h corresponds to a factor of 1. The boost factor is normally 2, but there is a possibility that it may have been limited below this value if it would otherwise cause the LED current to exceed the system maximum drive current (rather than that of the LED) of 5A.

GET_TOTAL_STATUS (01 48h) (-- n1 ... n33)
Also available at 200h for access when timers are running
This one actually calls GET_DIGITAL_STATUS, GET_ADC_ANALOG_STATUS and GET_USB_ANALOG_STATUS sequentially as a single command, in order to minimise USB traffic.

RESTORE_LEDS (01 4ch) (--)
Currently for testing only. It should restore all previously remembered configuration settings